

PATENT COOPERATION TREATY

PCT

NOTIFICATION OF ELECTION

(PCT Rule 61.2)

From the INTERNATIONAL BUREAU

To:

Commissioner
US Department of Commerce
United States Patent and Trademark
Office, PCT
2011 South Clark Place Room
CP2/5C24
Arlington, VA 22202
ETATS-UNIS D'AMERIQUE

in its capacity as elected Office

Date of mailing:

14 December 2000 (14.12.00)

International application No.:

PCT/DE00/01001

Applicant's or agent's file reference:

99P1974P

International filing date:

03 April 2000 (03.04.00)

Priority date:

02 June 1999 (02.06.99)

Applicant:

LIGGESMEYER, Peter

1. The designated Office is hereby notified of its election made:



in the demand filed with the International preliminary Examining Authority on:

29 August 2000 (29.08.00)



in a notice effecting later election filed with the International Bureau on:

2. The election ☒ was

was not

made before the expiration of 19 months from the priority date or, where Rule 32 applies, within the time limit under Rule 32.2(b).

The International Bureau of WIPO
34, chemin des Colombettes
1211 Geneva 20, Switzerland

Facsimile No.: (41-22) 740 14.35

Authorized officer:

J. Zahra

Telephone No.: (41-22) 338.83.38

PATENT COOPERATION TREATY

PCT

INTERNATIONAL PRELIMINARY EXAMINATION REPORT

(PCT Article 36 and Rule 70)

RECEIVED
MAY 17 2002
Technology Center 2100891980489
Translation

Applicant's or agent's file reference 99P1974P	FOR FURTHER ACTION See Notification of Transmittal of International Preliminary Examination Report (Form PCT/IPEA/416)	
International application No. PCT/DE00/01001	International filing date (day/month/year) 03 April 2000 (03.04.00)	Priority date (day/month/year) 02 June 1999 (02.06.99)
International Patent Classification (IPC) or national classification and IPC G06F 11/00		
Applicant SIEMENS AKTIENGESELLSCHAFT		

1. This international preliminary examination report has been prepared by this International Preliminary Examining Authority and is transmitted to the applicant according to Article 36.

2. This REPORT consists of a total of 5 sheets, including this cover sheet.

☒ This report is also accompanied by ANNEXES, i.e., sheets of the description, claims and/or drawings which have been amended and are the basis for this report and/or sheets containing rectifications made before this Authority (see Rule 70.16 and Section 607 of the Administrative Instructions under the PCT).

These annexes consist of a total of 5 sheets.

3. This report contains indications relating to the following items:

- I ☒ Basis of the report
- II ☐ Priority
- III ☐ Non-establishment of opinion with regard to novelty, inventive step and industrial applicability
- IV ☐ Lack of unity of invention
- V ☒ Reasoned statement under Article 35(2) with regard to novelty, inventive step or industrial applicability; citations and explanations supporting such statement
- VI ☐ Certain documents cited
- VII ☒ Certain defects in the international application
- VIII ☐ Certain observations on the international application

Date of submission of the demand 29 August 2000 (29.08.00)	Date of completion of this report 12 September 2001 (12.09.2001)
Name and mailing address of the IPEA/EP	Authorized officer
Facsimile No.	Telephone No.

INTERNATIONAL PRELIMINARY EXAMINATION REPORT

International application No.

PCT/DE00/01001

I. Basis of the report

1. With regard to the **elements** of the international application:*

- ☐ the international application as originally filed
- ☒ the description:
pages _____ 1-83 _____, as originally filed
pages _____, filed with the demand
pages _____, filed with the letter of _____
- ☒ the claims:
pages _____, as originally filed
pages _____, as amended (together with any statement under Article 19
pages _____, filed with the demand
pages _____ 1-16 _____, filed with the letter of _____ 020701
- ☒ the drawings:
pages _____ 1/12-12/12 _____, as originally filed
pages _____, filed with the demand
pages _____, filed with the letter of _____
- ☐ the sequence listing part of the description:
pages _____, as originally filed
pages _____, filed with the demand
pages _____, filed with the letter of _____

2. With regard to the **language**, all the elements marked above were available or furnished to this Authority in the language in which the international application was filed, unless otherwise indicated under this item.

These elements were available or furnished to this Authority in the following language _____ which is:

- ☐ the language of a translation furnished for the purposes of international search (under Rule 23.1(b)).
- ☐ the language of publication of the international application (under Rule 48.3(b)).
- ☐ the language of the translation furnished for the purposes of international preliminary examination (under Rule 55.2 and/or 55.3).

3. With regard to any **nucleotide and/or amino acid sequence** disclosed in the international application, the international preliminary examination was carried out on the basis of the sequence listing:

- ☐ contained in the international application in written form.
- ☐ filed together with the international application in computer readable form.
- ☐ furnished subsequently to this Authority in written form.
- ☐ furnished subsequently to this Authority in computer readable form.
- ☐ The statement that the subsequently furnished written sequence listing does not go beyond the disclosure in the international application as filed has been furnished.
- ☐ The statement that the information recorded in computer readable form is identical to the written sequence listing has been furnished.

4. ☐ The amendments have resulted in the cancellation of:

- ☐ the description, pages _____
- ☐ the claims, Nos. _____
- ☐ the drawings, sheets/fig _____

5. ☐ This report has been established as if (some of) the amendments had not been made, since they have been considered to go beyond the disclosure as filed, as indicated in the Supplemental Box (Rule 70.2(c)).**

* Replacement sheets which have been furnished to the receiving Office in response to an invitation under Article 14 are referred to in this report as "originally filed" and are not annexed to this report since they do not contain amendments (Rule 70.16 and 70.17).

** Any replacement sheet containing such amendments must be referred to under item 1 and annexed to this report.

INTERNATIONAL PRELIMINARY EXAMINATION REPORT

International application No.

PCT/DE 00/01001

V. Reasoned statement under Article 35(2) with regard to novelty, inventive step or industrial applicability; citations and explanations supporting such statement

1. Statement

Novelty (N)	Claims	1-16	YES
	Claims		NO
Inventive step (IS)	Claims	1-16	YES
	Claims		NO
Industrial applicability (IA)	Claims	1-16	YES
	Claims		NO

2. Citations and explanations

Reference is made to the following documents:

D1: Leveson N. et al., "Safety Verification of ADA Programs using Software Fault Trees", IEEE Software, US, Vol. 8, No. 4, July 1991, pp. 48-59, mentioned in the application.

D2: Vishnuvajjala R. et al., "Flow analysis for concurrent, reactive, real-time systems", Proceedings IEEE High-assurance engineering workshop, Canada, October 1996, pp. 176-183.

Document D1 is considered the prior art closest to the subject matter of independent Claims 1, 8, 15 and 16. It discloses a method for detecting a total error description of at least one part of a computer program, said total error description being detected whilst taking into account a control flow description.

The subject matter of Claim 1 and of Claims 8, 15 and 16 which concord therewith differs from D1 in that a control flow description and a data flow description are merged into a common flow description, and that the total error description is detected from element error descriptions,

wherein, in one structure of the total error description, a structure of the common flow description is taken into account.

The problem addressed by the present invention can therefore be considered that of producing a method that enables the total error description of one part of a computer program to be more reliably detected.

Although D1 contains isolated suggestions as to data errors, D1 does not contain any suggestion as to merging a control flow description and a data flow description into a common flow description.

Although it describes separate control flow descriptions and data flow descriptions, D2 does not contain any suggestion as to simultaneously taking into account control flow influences and data flow influences when detecting a total error.

Therefore, independent Claims 1, 8, 15 and 16 meet the requirements of novelty, inventive step and industrial applicability (PCT Article 33(2) to (4)).

Claims 2-7 are dependent on Claim 1 and Claims 9-14 are dependent on Claim 8 and therefore likewise meet the requirements of novelty, inventive step and industrial applicability.

VII. Certain defects in the international application

The following defects in the form or contents of the international application have been noted:

The third document cited in the description, by Liggesmeyer, is an entire book of more than 300 pages. The applicant should have indicated in the description the passages that are important for the application.

Contrary to PCT Rule 5.1(a)(iii), pages 3 to 5 of the description are not consistent with the claims.

Pages 20 to 82 of the description only contains a computer program with which the method as per the embodiment is carried out. It is therefore not clear what is meant by the sentence on page 19, lines 26-28, because the description does not show any further alternatives or possible uses.

VERTRAG ÜBER DIE INTERNATIONALE ZUSAMMENARBEIT AUF DEM
GEBIET DES PATENTWESENS

Absender: MIT DER INTERNATIONALEN VORLÄUFIGEN
PRÜFUNG BEAUFTRAGTE BEHÖRDE

An:

SIEMENS AKTIENGESELLSCHAFT

Postfach 22 16 34

D-80506 München

ALLEMAGNE

CT IPS AM Mch P/Ri

Eing. 14. Sep. 2001

GR

Frist

02.10.01

PCT

MITTEILUNG ÜBER DIE ÜBERSENDUNG
DES INTERNATIONALEN VORLÄUFIGEN
PRÜFUNGSBERICHTS

(Regel 71.1 PCT)

Absendedatum
(Tag/Monat/Jahr)

12.09.2001

Aktenzeichen des Anmelders oder Anwalts

199P01974WO

WICHTIGE MITTEILUNG

Internationales Aktenzeichen
PCT/DE00/01001

Internationales Anmeldedatum (Tag/Monat/Jahr)
03/04/2000

Prioritätsdatum (Tag/Monat/Jahr)
02/06/1999

Anmelder

SIEMENS AKTIENGESELLSCHAFT

1. Dem Anmelder wird mitgeteilt, daß ihm die mit der internationalen vorläufigen Prüfung beauftragte Behörde hiermit den zu der internationalen Anmeldung erstellten internationalen vorläufigen Prüfungsbericht; gegebenenfalls mit den dazugehörigen Anlagen, übermittelt.
2. Eine Kopie des Berichts wird - gegebenenfalls mit den dazugehörigen Anlagen - dem Internationalen Büro zur Weiterleitung an alle ausgewählten Ämter übermittelt.
3. Auf Wunsch eines ausgewählten Amtes wird das Internationale Büro eine Übersetzung des Berichts (jedoch nicht der Anlagen) ins Englische anfertigen und diesem Amt übermitteln.


4. ERINNERUNG

Zum Eintritt in die nationale Phase hat der Anmelder vor jedem ausgewählten Amt innerhalb von 30 Monaten ab dem Prioritätsdatum (oder in manchen Ämtern noch später) bestimmte Handlungen (Einreichung von Übersetzungen und Entrichtung nationaler Gebühren) vorzunehmen (Artikel 39 (1)) (siehe auch die durch das Internationale Büro im Formblatt PCT/IB/301 übermittelte Information).

Ist einem ausgewählten Amt eine Übersetzung der internationalen Anmeldung zu übermitteln, so muß diese Übersetzung auch Übersetzungen aller Anlagen zum internationalen vorläufigen Prüfungsbericht enthalten. Es ist Aufgabe des Anmelders, solche Übersetzungen anzufertigen und den betroffenen ausgewählten Ämtern direkt zuzuleiten.

Weitere Einzelheiten zu den maßgebenden Fristen und Erfordernissen der ausgewählten Ämter sind Band II des PCT-Leitfadens für Anmelder zu entnehmen.

Name und Postanschrift der mit der internationalen Prüfung beauftragten Behörde

 Europäisches Patentamt - P.B. 5818 Patentlaan 2
NL-2280 HV Rijswijk - Pays Bas
Tel. +31 70 340 - 2040 Tx: 31 651 epo nl
Fax: +31 70 340 - 3016

Bevollmächtigter Bediensteter

Cardenas, C

Tel. +31 70 340-3370




VERTRAG ÜBER DIE INTERNATIONALE ZUSAMMENARBEIT AUF DEM GEBIET DES PATENTWESENS

PCT

INTERNATIONALER VORLÄUFIGER PRÜFUNGSBERICHT

(Artikel 36 und Regel 70 PCT)

Aktenzeichen des Anmelders oder Anwalts 199P01974WO	WEITERES VORGEHEN siehe Mitteilung über die Übersendung des internationalen vorläufigen Prüfungsberichts (Formblatt PCT/IPEA/416)	
Internationales Aktenzeichen PCT/DE00/01001	Internationales Anmeldedatum (Tag/Monat/Jahr) 03/04/2000	Prioritätsdatum (Tag/Monat/Tag) 02/06/1999
Internationale Patentklassifikation (IPK) oder nationale Klassifikation und IPK G06F11/00		
Anmelder SIEMENS AKTIENGESELLSCHAFT		
<p>1. Dieser internationale vorläufige Prüfungsbericht wurde von der mit der internationalen vorläufigen Prüfung beauftragten Behörde erstellt und wird dem Anmelder gemäß Artikel 36 übermittelt.</p> <p>2. Dieser BERICHT umfaßt insgesamt 5 Blätter einschließlich dieses Deckblatts.</p> <p><input checked="" type="checkbox"/> Außerdem liegen dem Bericht ANLAGEN bei; dabei handelt es sich um Blätter mit Beschreibungen, Ansprüchen und/oder Zeichnungen, die geändert wurden und diesem Bericht zugrunde liegen, und/oder Blätter mit vor dieser Behörde vorgenommenen Berichtigungen (siehe Regel 70.16 und Abschnitt 607 der Verwaltungsrichtlinien zum PCT).</p> <p>Diese Anlagen umfassen insgesamt 5 Blätter.</p>		
<p>3. Dieser Bericht enthält Angaben zu folgenden Punkten:</p> <ul style="list-style-type: none">I <input checked="" type="checkbox"/> Grundlage des BerichtsII <input type="checkbox"/> PrioritätIII <input type="checkbox"/> Keine Erstellung eines Gutachtens über Neuheit, erfinderische Tätigkeit und gewerbliche AnwendbarkeitIV <input type="checkbox"/> Mangelnde Einheitlichkeit der ErfindungV <input checked="" type="checkbox"/> Begründete Feststellung nach Artikel 35(2) hinsichtlich der Neuheit, der erfinderischen Tätigkeit und der gewerblichen Anwendbarkeit; Unterlagen und Erklärungen zur Stützung dieser FeststellungVI <input type="checkbox"/> Bestimmte angeführte UnterlagenVII <input checked="" type="checkbox"/> Bestimmte Mängel der internationalen AnmeldungVIII <input type="checkbox"/> Bestimmte Bemerkungen zur internationalen Anmeldung		
Datum der Einreichung des Antrags 29/08/2000	Datum der Fertigstellung dieses Berichts 12.09.2001	
Name und Postanschrift der mit der internationalen vorläufigen Prüfung beauftragten Behörde:  Europäisches Patentamt - P.B. 5818 Patentlaan 2 NL-2280 HV Rijswijk - Pays Bas Tel. +31 70 340 - 2040 Tx: 31 651 epo nl Fax: +31 70 340 - 3016	Bevollmächtigter Bediensteter Herreman, G Tel. Nr. +31 70 340 3073	



I. Grundlage des Berichts

1. Hinsichtlich der **Bestandteile** der internationalen Anmeldung (*Ersatzblätter, die dem Anmeldeamt auf eine Aufforderung nach Artikel 14 hin vorgelegt wurden, gelten im Rahmen dieses Berichts als "ursprünglich eingereicht" und sind ihm nicht beigelegt, weil sie keine Änderungen enthalten (Regeln 70.16 und 70.17)*):
Beschreibung, Seiten:

1-83 ursprüngliche Fassung

Patentansprüche, Nr.:

1-16 eingegangen am 02/07/2001 mit Schreiben vom 29/06/2001

Zeichnungen, Blätter:

1/12-12/12 ursprüngliche Fassung

2. Hinsichtlich der **Sprache**: Alle vorstehend genannten Bestandteile standen der Behörde in der Sprache, in der die internationale Anmeldung eingereicht worden ist, zur Verfügung oder wurden in dieser eingereicht, sofern unter diesem Punkt nichts anderes angegeben ist.

Die Bestandteile standen der Behörde in der Sprache: zur Verfügung bzw. wurden in dieser Sprache eingereicht; dabei handelt es sich um

- ☐ die Sprache der Übersetzung, die für die Zwecke der internationalen Recherche eingereicht worden ist (nach Regel 23.1(b)).
 - ☐ die Veröffentlichungssprache der internationalen Anmeldung (nach Regel 48.3(b)).
 - ☐ die Sprache der Übersetzung, die für die Zwecke der internationalen vorläufigen Prüfung eingereicht worden ist (nach Regel 55.2 und/oder 55.3).
3. Hinsichtlich der in der internationalen Anmeldung offenbarten **Nucleotid- und/oder Aminosäuresequenz** ist die internationale vorläufige Prüfung auf der Grundlage des Sequenzprotokolls durchgeführt worden, das:
- ☐ in der internationalen Anmeldung in schriftlicher Form enthalten ist.
 - ☐ zusammen mit der internationalen Anmeldung in computerlesbarer Form eingereicht worden ist.
 - ☐ bei der Behörde nachträglich in schriftlicher Form eingereicht worden ist.
 - ☐ bei der Behörde nachträglich in computerlesbarer Form eingereicht worden ist.
 - ☐ Die Erklärung, daß das nachträglich eingereichte schriftliche Sequenzprotokoll nicht über den Offenbarungsgehalt der internationalen Anmeldung im Anmeldezeitpunkt hinausgeht, wurde vorgelegt.
 - ☐ Die Erklärung, daß die in computerlesbarer Form erfassten Informationen dem schriftlichen Sequenzprotokoll entsprechen, wurde vorgelegt.
4. Aufgrund der Änderungen sind folgende Unterlagen fortgefallen:

- ☐ Beschreibung, Seiten:
☐ Ansprüche, Nr.:
☐ Zeichnungen, Blatt:

5. ☐ Dieser Bericht ist ohne Berücksichtigung (von einigen) der Änderungen erstellt worden, da diese aus den angegebenen Gründen nach Auffassung der Behörde über den Offenbarungsgehalt in der ursprünglich eingereichten Fassung hinausgehen (Regel 70.2(c)).

(Auf Ersatzblätter, die solche Änderungen enthalten, ist unter Punkt 1 hinzuweisen; sie sind diesem Bericht beizufügen).

6. Etwaige zusätzliche Bemerkungen:

V. Begründete Feststellung nach Artikel 35(2) hinsichtlich der Neuheit, der erfinderischen Tätigkeit und der gewerblichen Anwendbarkeit; Unterlagen und Erklärungen zur Stützung dieser Feststellung

1. Feststellung

Neuheit (N)	Ja: Ansprüche	1-16
	Nein: Ansprüche	
Erfinderische Tätigkeit (ET)	Ja: Ansprüche	1-16
	Nein: Ansprüche	
Gewerbliche Anwendbarkeit (GA)	Ja: Ansprüche	1-16
	Nein: Ansprüche	

2. Unterlagen und Erklärungen
siehe Beiblatt

VII. Bestimmte Mängel der internationalen Anmeldung

Es wurde festgestellt, daß die internationale Anmeldung nach Form oder Inhalt folgende Mängel aufweist:
siehe Beiblatt

Zu Punkt V

Begründete Feststellung nach Artikel 35(2) hinsichtlich der Neuheit, der erfinderischen Tätigkeit und der gewerblichen Anwendbarkeit; Unterlagen und Erklärungen zur Stützung dieser Feststellung

Es wird auf die folgenden Dokumente verwiesen:

- D1: Leveson N. et al., "Safety Verification of ADA Programs using Software Fault Trees", IEEE Software, US, Bd. 8, Nr. 4, Juli 1991, S. 48-59, in der Anmeldung erwähnt.
- D2: Vishnuvajjala R. et al., "Flow analysis for concurrent, reactive, real-time systems", Proceedings IEEE High-assurance systems engineering workshop, Canada, Oktober 1996, S. 176-183.

Das Dokument D1 wird als nächstliegender Stand der Technik gegenüber dem Gegenstand der unabhängigen Ansprüche 1, 8, 15, 16 angesehen. Es offenbart ein Verfahren zur Ermittlung einer Gesamtfehlerbeschreibung zumindest eines Teils eines Computerprogramms, wobei die Gesamtfehlerbeschreibung unter Berücksichtigung einer Kontrollflußbeschreibung ermittelt wird.

Der Gegenstand des Anspruchs 1 und der damit übereinstimmenden Ansprüche 8, 15 und 16 unterscheidet sich dadurch von D1, daß eine Kontrollflußbeschreibung und eine Datenflußbeschreibung zu einer gemeinsamen Flußbeschreibung zusammengefaßt werden, sowie daß die Gesamtfehlerbeschreibung aus Elementenfehlerbeschreibungen ermittelt wird, wobei in einer Struktur der Gesamtfehlerbeschreibung eine Struktur der gemeinsamen Flußbeschreibung berücksichtigt wird.

Die mit der vorliegenden Erfindung zu lösende Aufgabe kann somit darin gesehen werden ein Verfahren zu schaffen um eine zuverlässigere Gesamtfehlerbeschreibung eines Teils eines Computerprogramms zu ermitteln.

Obwohl D1 isolierte Hinweise auf Datenfehler enthält, erhält man aus D1 keine Hinweise auf eine Zusammenfassung einer Kontrollflußbeschreibung und einer Datenflußbeschreibung zu einer gemeinsamen Flußbeschreibung.

Auch aus D2, obwohl es getrennt Kontrollflußbeschreibungen und Datenflußbeschreibungen nennt, erhält man keine Hinweise auf eine gleichzeitige Berücksichtigung von Kontrollflußfehlereinflüssen zusammen mit Datenflußfehlereinflüssen bei der Ermittlung eines Gesamtfehlers.

Deshalb erfüllen die unabhängigen Ansprüche 1, 8, 15, 16 die Erfordernisse hinsichtlich der Neuheit, der erfinderischen Tätigkeit und der gewerblichen Anwendbarkeit nach Artikel 33(2) bis (4) PCT.

Die Ansprüche 2-7 sind vom Anspruch 1 abhängig und die Ansprüche 9-14 sind vom Anspruch 8 abhängig und erfüllen damit ebenfalls die Erfordernisse des PCT bezüglich Neuheit, erfinderischen Tätigkeit und gewerblichen Anwendbarkeit.

Zu Punkt VII

Bestimmte Mängel der internationalen Anmeldung

Das dritte in der Beschreibung genannte Dokument von Liggesmeyer ist ein ganzes Buch von mehr als 300 Seiten. Die Anmelderin hätte die für die Anmeldung wichtigen Passagen in der Beschreibung nennen müssen.

Die Beschreibung auf Seiten 3 bis 5 steht nicht, wie in Regel 5.1 a) iii) PCT vorgeschrieben, in Einklang mit den Ansprüchen.

Die Beschreibung beinhaltet ab S. 20 bis S. 82 nur ein Computerprogramm, mit dem das Verfahren gemäß dem Ausführungsbeispiel realisiert ist. Es ist deshalb nicht klar, was mit dem Satz auf S. 19, Z. 26-28 gemeint ist, weil die Beschreibung keine weiteren Alternativen oder Anwendungsmöglichkeiten darstellt.

Patentansprüche

1. Verfahren zur Ermittlung einer Gesamtfehlerbeschreibung
zumindest eines Teils eines Computerprogramms, durch einen

5 Computer,

- bei dem zumindest der Teil des Computerprogramms gespeichert ist,
- bei dem eine Kontrollflußbeschreibung für den Teil des Computerprogramms, welche einen Fluß von Kontrollinformationen
10 in dem Teil des Computerprogramms beschreibt, und eine Datenflußbeschreibung für den Teil des Computerprogramms, welche einen Fluß von Daten in dem Teil des Computerprogramms beschreibt, ermittelt und in eine gemeinsame Flußbeschreibung für den Teil des Computerprogramms zusammengefaßt werden,
- bei dem Programmelemente aus dem Teil des Computerprogramms ausgewählt werden,
- bei dem für jedes ausgewählte Programmelement unter Verwendung einer gespeicherten Fehlerbeschreibung, die jeweils
20 einem Referenzelement zugeordnet ist, eine Elementenfehlerbeschreibung ermittelt wird, mit der mögliche Fehler des jeweiligen Programmelements beschrieben werden,
- bei dem mit einer Fehlerbeschreibung eines Referenzelements mögliche Fehler des jeweiligen Referenzelements beschrieben
25 werden, und
- bei dem aus den Elementenfehlerbeschreibungen die Gesamtfehlerbeschreibung ermittelt wird, wobei in einer Struktur der Gesamtfehlerbeschreibung eine Struktur der gemeinsamen Flußbeschreibung berücksichtigt wird.

30

2. Verfahren nach Anspruch 1,

bei dem die Kontrollflußbeschreibung in Form eines Kontrollflußgraphen vorliegt.

35 3. Verfahren nach Anspruch 1 oder 2,

bei dem die Datenflußbeschreibung in Form eines Datenflußgraphen vorliegt.

4. Verfahren nach einem der vorangegangenen Ansprüche,
bei dem die Fehlerbeschreibung in Form eines gespeicherten
Fehlerbaums vorliegt,
- 5 • bei dem die Elementenfehlerbeschreibung als Elementenfeh-
lerbaum ermittelt wird, und
- bei dem die Gesamtfehlerbeschreibung als Gesamtfehlerbaum
ermittelt wird.
- 10 5. Verfahren nach einem der vorangegangenen Ansprüche,
eingesetzt zur Fehleranalyse des Teils des Computerprogramms.
6. Verfahren nach einem der vorangegangenen Ansprüche,
- bei dem die Gesamtfehlerbeschreibung als Gesamtfehlerbaum
15 ermittelt wird,
- bei dem der Gesamtfehlerbaum verändert wird hinsichtlich
vorgegebener Rahmenbedingungen.
7. Verfahren nach Anspruch 6,
- 20 bei dem die Veränderung durch Hinzufügen eines Ergänzungsfeh-
lerbaums erfolgt.
8. Anordnung zur Ermittlung einer Gesamtfehlerbeschreibung
zumindest eines Teils eines Computerprogramms, durch einen
25 Computer,
- mit einem Prozessor, der derart eingerichtet ist, daß folgen-
de Schritte durchführbar sind:
- zumindest der Teil des Computerprogramms ist gespeichert,
- es wird eine Kontrollflußbeschreibung für den Teil des Com-
puterprogramms, welche einen Fluß von Kontrollinformationen
30 in dem Teil des Computerprogramms beschreibt, ermittelt und
es wird eine Datenflußbeschreibung für den Teil des Compu-
terprogramms, welche einen Fluß von Daten in dem Teil des
Computerprogramms beschreibt, ermittelt und es werden die
35 Kontrollflußbeschreibung und die Datenflußbeschreibung in
eine gemeinsame Flußbeschreibung für den Teil des Compu-
terprogramms zusammengefaßt,

Programmelemente werden aus dem Teil des Computerprogramms ausgewählt,

- für jedes ausgewählte Programmelement wird unter Verwendung einer gespeicherten Fehlerbeschreibung, die jeweils einem Referenzelement zugeordnet ist, eine Elementenfehlerbeschreibung ermittelt, mit der mögliche Fehler des jeweiligen Programmelements beschrieben werden,
- mit einer Fehlerbeschreibung eines Referenzelements werden mögliche Fehler des jeweiligen Referenzelements beschrieben, und
- aus den Elementenfehlerbeschreibungen wird die Gesamtfehlerbeschreibung ermittelt, wobei in einer Struktur der Gesamtfehlerbeschreibung eine Struktur der gemeinsamen Flußbeschreibung berücksichtigt wird.

9. Anordnung nach Anspruch 8,

bei der der Prozessor derart eingerichtet ist, daß die Kontrollflußbeschreibung in Form eines Kontrollflußgraphen vorliegt.

10. Anordnung nach Anspruch 8 oder 9,

bei der der Prozessor derart eingerichtet ist, daß die Datenflußbeschreibung in Form eines Datenflußgraphen vorliegt.

11. Anordnung nach einem der Ansprüche 8 bis 10,

bei der der Prozessor derart eingerichtet ist, daß

- die Fehlerbeschreibung in Form eines gespeicherten Fehlerbaums vorliegt,
- die Elementenfehlerbeschreibung als Elementenfehlerbaum ermittelt wird, und
- die Gesamtfehlerbeschreibung als Gesamtfehlerbaum ermittelt wird.

12. Anordnung nach einem der Ansprüche 8 bis 11,

eingesetzt zur Fehleranalyse des Teils des Computerprogramms.

13. Anordnung nach einem der Ansprüche 8 bis 12,

bei der der Prozessor derart eingerichtet ist, daß

- die Gesamtfehlerbeschreibung als Gesamtfehlerbaum ermittelt wird,
- der Gesamtfehlerbaum verändert wird hinsichtlich vorgegebbarer Rahmenbedingungen.

14. Anordnung nach Anspruch 13,

bei der der Prozessor derart eingerichtet ist, daß die Veränderung durch Hinzufügen eines Ergänzungsfehlerbaums erfolgt.

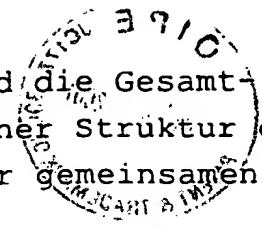
15. Computerprogramm-Erzeugnis, das ein computerlesbares Speichermedium umfaßt, auf dem ein Programm gespeichert ist,

das es einem Computer ermöglicht, nachdem es in einen Speicher des Computers geladen worden ist, folgende Schritte

durchzuführen zur Ermittlung einer Gesamtfehlerbeschreibung zumindest eines Teils eines Computerprogramms:

- zumindest der Teil des Computerprogramms ist gespeichert,
- es wird eine Kontrollflußbeschreibung für den Teil des Computerprogramms, welche eine Fluß von Kontrollinformationen in dem Teil des Computerprogramms beschreibt, ermittelt und es wird eine Datenflußbeschreibung für den Teil des Computerprogramms, welche einen Fluß von Daten in dem Teil des Computerprogramms beschreibt, ermittelt und es werden die Kontrollflußbeschreibung und die Datenflußbeschreibung in eine gemeinsame Flußbeschreibung für den Teil des Computerprogramms zusammengefaßt,
- Programmelemente werden aus dem Teil des Computerprogramms ausgewählt,
- für jedes ausgewählte Programmelement wird unter Verwendung einer gespeicherten Fehlerbeschreibung, die jeweils einem Referenzelement zugeordnet ist, eine Elementenfehlerbeschreibung ermittelt, mit der mögliche Fehler des jeweiligen Programmelements beschrieben werden,
- mit einer Fehlerbeschreibung eines Referenzelements werden mögliche Fehler des jeweiligen Referenzelements beschrieben, und

aus den Elementenfehlerbeschreibungen wird die Gesamtfehlerbeschreibung ermittelt, wobei in einer Struktur der Gesamtfehlerbeschreibung eine Struktur der gemeinsamen Flußbeschreibung berücksichtigt wird.



- 5
16. Computerlesbares Speichermedium, auf dem ein Programm gespeichert ist, das es einem Computer ermöglicht, nachdem es in einen Speicher des Computers geladen worden ist, folgende Schritte durchzuführen zur Ermittlung einer Gesamtfehlerbeschreibung zumindest eines Teils eines Computerprogramms:
- 10
- zumindest der Teil des Computerprogramms ist gespeichert,
 - es wird eine Kontrollflußbeschreibung für den Teil des Computerprogramms, welche eine Fluß von Kontrollinformationen in dem Teil des Computerprograms beschreibt, ermittelt und
 - 15 es wird eine Datenflußbeschreibung für den Teil des Computerprogramms, welche einen Fluß von Daten in dem Teil des Computerprograms beschreibt, ermittelt und es werden die Kontrollflußbeschreibung und die Datenflußbeschreibung in eine gemeinsame Flußbeschreibung für den Teil des Computerprogramms zusammengefaßt,
 - 20
 - Programmelemente werden aus dem Teil des Computerprogramms ausgewählt,
 - für jedes ausgewählte Programmelement wird unter Verwendung einer gespeicherten Fehlerbeschreibung, die jeweils einem
 - 25 Referenzelement zugeordnet ist, eine Elementenfehlerbeschreibung ermittelt, mit der mögliche Fehler des jeweiligen Programmelements beschrieben werden,
 - mit einer Fehlerbeschreibung eines Referenzelements werden mögliche Fehler des jeweiligen Referenzelements beschrieben, und
 - 30
 - aus den Elementenfehlerbeschreibungen wird die Gesamtfehlerbeschreibung ermittelt, wobei in einer Struktur der Gesamtfehlerbeschreibung eine Struktur der gemeinsamen Flußbeschreibung berücksichtigt wird.
 - 35

VERTRAG ÜBER DIE INTERNATIONALE ZUSAMMENARBEIT AUF DEM GEBIET DES PATENTWESENS

Absender: INTERNATIONALE RECHERCHENBEHÖRDE

PCT

MITTEILUNG ÜBER DIE ÜBERMITTLUNG DES
INTERNATIONALEN RECHERCHENBERICHTS
ODER DER ERKLÄRUNG

An

SIEMENS AKTIENGESELLSCHAFT
Postfach 22 16 34
D-80506 München
GERMANY

ZT GG VM Mch P/Ri

Eing. 25. Aug. 2000

GR
Frist

(Regel 44.1 PCT)

*Erwartete Brief d. Erklärt trifft die Einfuhr
nicht, Abkantung geben 28.8. 87*

Absenddatum
(Tag/Monat/Jahr)

28/08/2000

Aktenzeichen des Anmelders oder Anwalts

99P1974P

WEITERES VORGEHEN

siehe Punkte 1 und 4 unten

Internationales Aktenzeichen

PCT/DE 00/01001

Internationales Anmeldedatum

(Tag/Monat/Jahr)

03/04/2000

Anmelder

SIEMENS AKTIENGESELLSCHAFT

1. ☒ Dem Anmelder wird mitgeteilt, daß der internationale Recherchenbericht erstellt wurde und ihm hiermit übermittelt wird.

Einreichung von Änderungen und einer Erklärung nach Artikel 19:

Der Anmelder kann auf eigenen Wunsch die Ansprüche der internationalen Anmeldung ändern (siehe Regel 46):

Bis wann sind Änderungen einzureichen?

Die Frist zur Einreichung solcher Änderungen beträgt üblicherweise zwei Monate ab der Übermittlung des internationalen Recherchenberichts; weitere Einzelheiten sind den Anmerkungen auf dem Beiblatt zu entnehmen.

Wo sind Änderungen einzureichen?

Unmittelbar beim Internationalen Büro der WIPO, 34, CHEMIN des Colombettes, CH-1211 Genf 20,
Telefaxnr.: (41-22) 740.14.35

Nähere Hinweise sind den Anmerkungen auf dem Beiblatt zu entnehmen.

2. ☐ Dem Anmelder wird mitgeteilt, daß kein internationaler Recherchenbericht erstellt wird und daß ihm hiermit die Erklärung nach Artikel 17(2)a) übermittelt wird.
3. ☐ Hinsichtlich des Widerspruchs gegen die Entrichtung einer zusätzlichen Gebühr (zusätzlicher Gebühren) nach Regel 40.2 wird dem Anmelder mitgeteilt, daß

☐ der Widerspruch und die Entscheidung hierüber zusammen mit seinem Antrag auf Übermittlung des Wortlauts sowohl des Widerspruchs als auch der Entscheidung hierüber an die Bestimmungsbüro dem Internationalen Büro übermittelt worden sind.

☐ noch keine Entscheidung über den Widerspruch vorliegt; der Anmelder wird benachrichtigt, sobald eine Entscheidung getroffen wurde.

4. **Weiteres Vorgehen:** Der Anmelder wird auf folgendes aufmerksam gemacht:

Kurz nach Ablauf von **18 Monaten** seit dem Prioritätsdatum wird die internationale Anmeldung vom Internationalen Büro veröffentlicht. Will der Anmelder die Veröffentlichung verhindern oder auf einen späteren Zeitpunkt verschieben, so muß gemäß Regel 90^{bis} bzw. 90^{ter} vor Abschluß der technischen Vorbereitungen für die internationale Veröffentlichung eine Erklärung über die Zurücknahme der internationalen Anmeldung oder des Prioritätsanspruchs beim Internationalen Büro eingehen.

Innerhalb von **19 Monaten** seit dem Prioritätsdatum ist ein Antrag auf internationale vorläufige Prüfung einzureichen, wenn der Anmelder den Eintritt in die nationale Phase bis zu 30 Monaten seit dem Prioritätsdatum (in manchen Ämtern sogar noch länger) verschieben möchte.

Innerhalb von **20 Monaten** seit dem Prioritätsdatum muß der Anmelder die für den Eintritt in die nationale Phase vorgeschriebenen Handlungen vor allen Bestimmungsbüro vornehmen, die nicht innerhalb von 19 Monaten seit dem Prioritätsdatum in der Anmeldung oder einer nachträglichen Auswählerklärung ausgewählt wurden oder nicht ausgewählt werden konnten, da für sie Kapitel II des Vertrages nicht verbindlich ist.

Name und Postanschrift der Internationalen Recherchenbehörde



Europäisches Patentamt, P.B. 5818 Patentlaan 2
NL-2280 HV Rijswijk
Tel. (+31-70) 340-2040, Tx. 31 651 epo nl,
Fax: (+31-70) 340-3016

Bevollmächtigter Bediensteter

Lucia Van Pinxteren

ANMERKUNGEN ZU FORMBLATT PCT/ISA/220

Diese Anmerkungen sollen grundlegende Hinweise zur Einreichung von Änderungen gemäß Artikel 19 geben. Diesen Anmerkungen liegen die Erfordernisse des Vertrags über die internationale Zusammenarbeit auf dem Gebiet des Patentwesens (PCT), der Ausführungsordnung und der Verwaltungsrichtlinien zu diesem Vertrag zugrunde. Bei Abweichungen zwischen diesen Anmerkungen und obengenannten Texten sind letztere maßgebend. Nähere Einzelheiten sind dem PCT-Leitfaden für Anmelder, einer Veröffentlichung der WIPO, zu entnehmen.
Die in diesen Anmerkungen verwendeten Begriffe "Artikel", "Regel" und "Abschnitt" beziehen sich jeweils auf die Bestimmungen des PCT-Vertrags, der PCT-Ausführungsordnung bzw. der PCT-Verwaltungsrichtlinien.

HINWEISE ZU ÄNDERUNGEN GEMÄSS ARTIKEL 19

Nach Erhalt des internationalen Recherchenberichts hat der Anmelder die Möglichkeit, einmal die Ansprüche der internationalen Anmeldung zu ändern. Es ist jedoch zu betonen, daß, da alle Teile der internationalen Anmeldung (Ansprüche, Beschreibung und Zeichnungen) während des internationalen vorläufigen Prüfungsverfahrens geändert werden können, normalerweise keine Notwendigkeit besteht, Änderungen der Ansprüche nach Artikel 19 einzureichen, außer wenn der Anmelder z.B. zum Zwecke eines vorläufigen Schutzes die Veröffentlichung dieser Ansprüche wünscht oder ein anderer Grund für eine Änderung der Ansprüche vor ihrer internationalen Veröffentlichung vorliegt. Weiterhin ist zu beachten, daß ein vorläufiger Schutz nur in einigen Staaten erhältlich ist.

Welche Teile der internationalen Anmeldung können geändert werden?

Im Rahmen von Artikel 19 können nur die Ansprüche geändert werden.

In der internationalen Phase können die Ansprüche auch nach Artikel 34 vor der mit der internationalen vorläufigen Prüfung beauftragten Behörde geändert (oder nochmals geändert) werden. Die Beschreibung und die Zeichnungen können nur nach Artikel 34 vor der mit der internationalen vorläufigen Prüfung beauftragten Behörde geändert werden.

Beim Eintritt in die nationale Phase können alle Teile der internationalen Anmeldung nach Artikel 28 oder gegebenenfalls Artikel 41 geändert werden.

Bis wann sind Änderungen einzureichen?

Innerhalb von zwei Monaten ab der Übermittlung des internationalen Recherchenberichts oder innerhalb von sechzehn Monaten ab dem Prioritätsdatum, je nachdem, welche Frist später abläuft. Die Änderungen gelten jedoch als rechtzeitig eingereicht, wenn sie dem Internationalen Büro nach Ablauf der maßgebenden Frist, aber noch vor Abschluß der technischen Vorbereitungen für die internationale Veröffentlichung (Regel 46.1) zu gehen.

Wo sind die Änderungen nicht einzureichen?

Die Änderungen können nur beim Internationalen Büro, nicht aber beim Anmeldeamt oder der Internationalen Recherchenbehörde eingereicht werden (Regel 46.2).

Falls ein Antrag auf internationale vorläufige Prüfung eingereicht wurde/wird, siehe unten.

In welcher Form können Änderungen erfolgen?

Eine Änderung kann erfolgen durch Streichung eines oder mehrerer ganzer Ansprüche, durch Hinzufügung eines oder mehrerer neuer Ansprüche oder durch Änderung des Wortlauts eines oder mehrerer Ansprüche in der eingereichten Fassung.

Für jedes Anspruchsblatt, das sich aufgrund einer oder mehrerer Änderungen von dem ursprünglich eingereichten Blatt unterscheidet, ist ein Ersatzblatt einzureichen.

Alle Ansprüche, die auf einem Ersatzblatt erscheinen, sind mit arabischen Ziffern zu numerieren. Wird ein Anspruch gestrichen, so brauchen die anderen Ansprüche nicht neu nummeriert zu werden. Im Fall einer Neunummerierung sind die Ansprüche fortlaufend zu numerieren (Verwaltungsrichtlinien, Abschnitt 205 b)).

Die Änderungen sind in der Sprache abzufassen, in der die internationale Anmeldung veröffentlicht wird.

Welche Unterlagen sind den Änderungen beizufügen?

Begleitschreiben (Abschnitt 205 b)):

Die Änderungen sind mit einem Begleitschreiben einzureichen.

Das Begleitschreiben wird nicht zusammen mit der internationalen Anmeldung und den geänderten Ansprüchen veröffentlicht. Es ist nicht zu verwechseln mit der "Erklärung nach Artikel 19(1)" (siehe unten, "Erklärung nach Artikel 19(1)").

Das Begleitschreiben ist nach Wahl des Anmelders in englischer oder französischer Sprache abzufassen. Bei englischsprachigen internationalen Anmeldungen ist das Begleitschreiben aber ebenfalls in englischer, bei französischsprachigen internationalen Anmeldungen in französischer Sprache abzufassen.

ANMERKUNGEN ZU FORMBLATT PCT/ISA/220 (F r t s t z u n g)

Im Begleitschreiben sind die Unterschiede zwischen den Ansprüchen in der eingereichten Fassung und den geänderten Ansprüchen anzugeben. So ist insbesondere zu jedem Anspruch in der internationalen Anmeldung anzugeben (gleichlautende Angaben zu verschiedenen Ansprüchen können zusammengefaßt werden), ob

- i) der Anspruch unverändert ist;
- ii) der Anspruch gestrichen worden ist;
- iii) der Anspruch neu ist;
- iv) der Anspruch einen oder mehrere Ansprüche in der eingereichten Fassung ersetzt;
- v) der Anspruch auf die Teilung eines Anspruchs in der eingereichten Fassung zurückzuführen ist.

Im folgenden sind Beispiele angegeben, wie Änderungen im Begleitschreiben zu erläutern sind:

1. [Wenn anstelle von ursprünglich 48 Ansprüchen nach der Änderung einiger Ansprüche 51 Ansprüche existieren]:
"Die Ansprüche 1 bis 29, 31, 32, 34, 35, 37 bis 48 werden durch geänderte Ansprüche gleicher Numerierung ersetzt; Ansprüche 30, 33 und 36 unverändert; neue Ansprüche 49 bis 51 hinzugefügt."
2. [Wenn anstelle von ursprünglich 15 Ansprüchen nach der Änderung aller Ansprüche 11 Ansprüche existieren]:
"Geänderte Ansprüche 1 bis 11 treten an die Stelle der Ansprüche 1 bis 15."
3. [Wenn ursprünglich 14 Ansprüche existierten und die Änderungen darin bestehen, daß einige Ansprüche gestrichen werden und neue Ansprüche hinzugefügt werden]:
"Ansprüche 1 bis 6 und 14 unverändert; Ansprüche 7 bis 13 gestrichen; neue Ansprüche 15, 16 und 17 hinzugefügt. "Oder" Ansprüche 7 bis 13 gestrichen; neue Ansprüche 15, 16 und 17 hinzugefügt; alle übrigen Ansprüche unverändert."
4. [Wenn verschiedene Arten von Änderungen durchgeführt werden]:
"Ansprüche 1-10 unverändert; Ansprüche 11 bis 13, 18 und 19 gestrichen; Ansprüche 14, 15 und 16 durch geänderten Anspruch 14 ersetzt; Anspruch 17 in geänderte Ansprüche 15, 16 und 17 unterteilt; neue Ansprüche 20 und 21 hinzugefügt."

"Erklärung nach Artikel 19(1)" (Regel 46.4)

Den Änderungen kann eine Erklärung beigefügt werden, mit der die Änderungen erläutert und ihre Auswirkungen auf die Beschreibung und die Zeichnungen dargelegt werden (die nicht nach Artikel 19 (1) geändert werden können).

Die Erklärung wird zusammen mit der internationalen Anmeldung und den geänderten Ansprüchen veröffentlicht.

Sie ist in der Sprache abzufassen, in der die internationale Anmeldung veröffentlicht wird.

Sie muß kurz gehalten sein und darf, wenn in englischer Sprache abgefaßt oder ins Englische übersetzt, nicht mehr als 500 Wörter umfassen.

Die Erklärung ist nicht zu verwechseln mit dem Begleitschreiben, das auf die Unterschiede zwischen den Ansprüchen in der eingereichten Fassung und den geänderten Ansprüchen hinweist, und ersetzt letzteres nicht. Sie ist auf einem gesonderten Blatt einzureichen und in der Überschrift als solche zu kennzeichnen, vorzugsweise mit den Worten "Erklärung nach Artikel 19 (1)".

Die Erklärung darf keine herabsetzenden Äußerungen über den internationalen Recherchenbericht oder die Bedeutung von in dem Bericht angeführten Veröffentlichungen enthalten. Sie darf auf im internationalen Recherchenbericht angeführte Veröffentlichungen, die sich auf einen bestimmten Anspruch beziehen, nur im Zusammenhang mit einer Änderung dieses Anspruchs Bezug nehmen.

Auswirkungen eines bereits gestellten Antrags auf internationale vorläufige Prüfung

Ist zum Zeitpunkt der Einreichung von Änderungen nach Artikel 19 bereits ein Antrag auf internationale vorläufige Prüfung gestellt worden, so sollte der Anmelder in seinem Interesse gleichzeitig mit der Einreichung der Änderungen beim Internationalen Büro auch eine Kopie der Änderungen bei der mit der internationalen vorläufigen Prüfung beauftragten Behörde einreichen (siehe Regel 62.2 a), erster Satz).

Auswirkungen von Änderungen hinsichtlich der Übersetzung der internationalen Anmeldung beim Eintritt in die nationale Phase

Der Anmelder wird darauf hingewiesen, daß bei Eintritt in die nationale Phase möglicherweise anstatt oder zusätzlich zu der Übersetzung der Ansprüche in der eingereichten Fassung eine Übersetzung der nach Artikel 19 geänderten Ansprüche an die bestimmten/ausgewählten Ämter zu übermitteln ist.

Nähere Einzelheiten über die Erfordernisse jedes bestimmten/ausgewählten Amtes sind Band II des PCT-Leitfadens für Anmelder zu entnehmen.

**VERTRAG ÜBER DIE INTERNATIONALE ZUSAMMENARBEIT
AUF DEM GEBIET DES PATENTWESENS**

PCT

INTERNATIONALER RECHERCHENBERICHT

(Artikel 18 sowie Regeln 43 und 44 PCT)

Aktenzeichen des Anmelders oder Anwalts 99P1974P	WEITERES VORGEHEN siehe Mitteilung über die Übermittlung des internationalen Recherchenberichts (Formblatt PCT/ISA/220) sowie, soweit zutreffend, nachstehender Punkt 5	
Internationales Aktenzeichen PCT/DE 00/ 01001	Internationales Anmeldedatum (Tag/Monat/Jahr) 03/04/2000	(Frühestes) Prioritätsdatum (Tag/Monat/Jahr) 02/06/1999
Anmelder SIEMENS AKTIENGESELLSCHAFT		

Dieser internationale Recherchenbericht wurde von der Internationalen Recherchenbehörde erstellt und wird dem Anmelder gemäß Artikel 18 übermittelt. Eine Kopie wird dem Internationalen Büro übermittelt.

Dieser internationale Recherchenbericht umfaßt insgesamt 3 Blätter.

☒ Darüber hinaus liegt ihm jeweils eine Kopie der in diesem Bericht genannten Unterlagen zum Stand der Technik bei.

1. Grundlage des Berichts

- a. Hinsichtlich der **Sprache** ist die internationale Recherche auf der Grundlage der internationalen Anmeldung in der Sprache durchgeführt worden, in der sie eingereicht wurde, sofern unter diesem Punkt nichts anderes angegeben ist.

☐ Die internationale Recherche ist auf der Grundlage einer bei der Behörde eingereichten Übersetzung der internationalen Anmeldung (Regel 23.1 b)) durchgeführt worden.

- b. Hinsichtlich der in der internationalen Anmeldung offenbarten **Nucleotid- und/oder Aminosäuresequenz** ist die internationale Recherche auf der Grundlage des Sequenzprotokolls durchgeführt worden, das

☐ in der internationalen Anmeldung in schriftlicher Form enthalten ist.

☐ zusammen mit der internationalen Anmeldung in computerlesbarer Form eingereicht worden ist.

☐ bei der Behörde nachträglich in schriftlicher Form eingereicht worden ist.

☐ bei der Behörde nachträglich in computerlesbarer Form eingereicht worden ist.

☐ Die Erklärung, daß das nachträglich eingereichte schriftliche Sequenzprotokoll nicht über den Offenbarungsgehalt der internationalen Anmeldung im Anmeldezeitpunkt hinausgeht, wurde vorgelegt.

☐ Die Erklärung, daß die in computerlesbarer Form erfaßten Informationen dem schriftlichen Sequenzprotokoll entsprechen, wurde vorgelegt.

2. ☐ **Bestimmte Ansprüche haben sich als nicht recherchierbar erwiesen** (siehe Feld I).

3. ☐ **Mangelnde Einheitlichkeit der Erfindung** (siehe Feld II).

4. Hinsichtlich der Bezeichnung der Erfindung

☒ wird der vom Anmelder eingereichte Wortlaut genehmigt.

☐ wurde der Wortlaut von der Behörde wie folgt festgesetzt:

5. Hinsichtlich der Zusammenfassung

☒ wird der vom Anmelder eingereichte Wortlaut genehmigt.

☐ wurde der Wortlaut nach Regel 38.2b) in der in Feld III angegebenen Fassung von der Behörde festgesetzt. Der Anmelder kann der Behörde innerhalb eines Monats nach dem Datum der Absendung dieses internationalen Recherchenberichts eine Stellungnahme vorlegen.

6. Folgende Abbildung der **Zeichnungen** ist mit der Zusammenfassung zu veröffentlichen: Abb. Nr. 2

☒ wie vom Anmelder vorgeschlagen

☐ weil der Anmelder selbst keine Abbildung vorgeschlagen hat.

☐ weil diese Abbildung die Erfindung besser kennzeichnet.

☐ keine der Abb.

A. KLASSIFIZIERUNG DES ANMELDUNGSGEGENSTANDES
IPK 7 G06F11/00

Nach der Internationalen Patentklassifikation (IPK) oder nach der nationalen Klassifikation und der IPK

B. RECHERCHIERTE GEBIETE

Recherchierter Mindestprüfstoff (Klassifikationssystem und Klassifikationssymbole)
IPK 7 G06F

Recherchierte aber nicht zum Mindestprüfstoff gehörende Veröffentlichungen, soweit diese unter die recherchierten Gebiete fallen

Während der internationalen Recherche konsultierte elektronische Datenbank (Name der Datenbank und evtl. verwendete Suchbegriffe)

INSPEC, IBM-TDB, EPO-Internal, PAJ, WPI Data

C. ALS WESENTLICH ANGESEHENE UNTERLAGEN

Kategorie*	Bezeichnung der Veröffentlichung, soweit erforderlich unter Angabe der in Betracht kommenden Teile	Betr. Anspruch Nr.
X	LEVESON N G ET AL: "SAFETY VERIFICATION OF ADA PROGRAMS USING SOFTWARE FAULT TREES" IEEE SOFTWARE,US,IEEE COMPUTER SOCIETY. LOS ALAMITOS, Bd. 8, Nr. 4, Juli 1991 (1991-07), Seiten 48-59, XP000290816 ISSN: 0740-7459 in der Anmeldung erwähnt	1,2,4,5, 8,9,11, 12,15,16
Y	das ganze Dokument --- -/--	3,10



Weitere Veröffentlichungen sind der Fortsetzung von Feld C zu entnehmen



Siehe Anhang Patentfamilie

* Besondere Kategorien von angegebenen Veröffentlichungen :

"A" Veröffentlichung, die den allgemeinen Stand der Technik definiert, aber nicht als besonders bedeutsam anzusehen ist

"E" älteres Dokument, das jedoch erst am oder nach dem internationalen Anmeldedatum veröffentlicht worden ist

"L" Veröffentlichung, die geeignet ist, einen Prioritätsanspruch zweifelhaft erscheinen zu lassen, oder durch die das Veröffentlichungsdatum einer anderen im Recherchenbericht genannten Veröffentlichung belegt werden soll oder die aus einem anderen besonderen Grund angegeben ist (wie ausgeführt)

"O" Veröffentlichung, die sich auf eine mündliche Offenbarung, eine Benutzung, eine Ausstellung oder andere Maßnahmen bezieht

"P" Veröffentlichung, die vor dem internationalen Anmeldedatum, aber nach dem beanspruchten Prioritätsdatum veröffentlicht worden ist

"T" Spätere Veröffentlichung, die nach dem internationalen Anmeldedatum oder dem Prioritätsdatum veröffentlicht worden ist und mit der Anmeldung nicht kollidiert, sondern nur zum Verständnis des der Erfindung zugrundeliegenden Prinzips oder der ihr zugrundeliegenden Theorie angegeben ist

"X" Veröffentlichung von besonderer Bedeutung; die beanspruchte Erfindung kann allein aufgrund dieser Veröffentlichung nicht als neu oder auf erfindnerischer Tätigkeit beruhend betrachtet werden

"Y" Veröffentlichung von besonderer Bedeutung; die beanspruchte Erfindung kann nicht als auf erfindnerischer Tätigkeit beruhend betrachtet werden, wenn die Veröffentlichung mit einer oder mehreren anderen Veröffentlichungen dieser Kategorie in Verbindung gebracht wird und diese Verbindung für einen Fachmann naheliegend ist

"&" Veröffentlichung, die Mitglied derselben Patentfamilie ist

Datum des Abschlusses der internationalen Recherche

9. August 2000

Absendedatum des internationalen Recherchenberichts

28/08/2000

Name und Postanschrift der Internationalen Recherchenbehörde
Europäisches Patentamt, P.B. 5818 Patentlaan 2
NL - 2280 HV Rijswijk
Tel. (+31-70) 340-2040, Tx. 31 651 epo nl,
Fax: (+31-70) 340-3016

Bevollmächtigter Bediensteter

Herreman, G

C.(Fortsetzung) ALS WESENTLICH ANGESEHENE UNTERLAGEN

Kategorie°	Bezeichnung der Veröffentlichung, soweit erforderlich unter Angabe der in Betracht kommenden Teile	Betr. Anspruch Nr.
Y	VISHNUVAJJALA R ET AL: "Flow analysis for concurrent, reactive, real-time systems" PROCEEDINGS. IEEE HIGH-ASSURANCE SYSTEMS ENGINEERING WORKSHOP (CAT. NO.96TB100076), NIAGARA ON THE LAKE, ONT., CANADA, 21-22 OCT. 1996, Seiten 176-183, XP002144628 1997, Los Alamitos, CA, USA, IEEE Comput. Soc. Press, USA ISBN: 0-8186-7629-9	3,10
A	Zusammenfassung Seite 176, linke Spalte, Zeile 1 -rechte Spalte, Zeile 26 Seite 177, linke Spalte, Zeile 31 - Zeile 38	1,2,4-9, 11-16
A	----- IEEE TRANSACTIONS ON SOFTWARE ENGINEERING., Bd. SE-10, Nr. 4, Juli 1984 (1984-07), Seiten 352-357, XP002144629 IEEE INC. NEW YORK., US ISSN: 0098-5589 in der Anmeldung erwähnt -----	

PCT-ANTRAG

Original (für EINREICHUNG) - gedruckt am 13.03.2000 03:04:30 PM

0	Vom Anmeldeamt auszufüllen	
0-1	Internationales Aktenzeichen.	
0-2	Internationales Anmeldedatum	
0-3	Name des Anmeldeamts und "PCT International Application"	
0-4 0-4-1	Formular - PCT/RO/101 PCT-Antrag erstellt durch Benutzung von	PCT-EASY Version 2.90 (aktualisiert 08.03.2000)
0-5	Antragssuchen Der Unterzeichnete beantragt, daß die vorliegende internationale Anmeldung nach dem Vertrag über die internationale Zusammenarbeit auf dem Gebiet des Patentwesens behandelt wird	
0-6	(Vom Anmelder gewähltes) Anmeldeamt	Deutsches Patent- und Markenamt (RO/DE)
0-7	Aktenzeichen des Anmelders oder Anwalts	99P1974P
I	Bezeichnung der Erfindung	VERFAHREN UND ANORDNUNG ZUR ERMITTLUNG EINER GESAMTFEHLERBESCHREIBUNG ZUMINDEST EINES TEILS EINES COMPUTERPROGRAMMS SOWIE COMPUTERPROGRAMM-ERZEUGNIS UND COMPUTERLESBARES SPEICHERMEDIUM
II	Anmelder	nur Anmelder
II-1	Diese Person ist	Alle Bestimmungsstaaten mit Ausnahme von US
II-2	Anmelder für	SIEMENS AKTIENGESELLSCHAFT
II-4	Name	Wittelsbacherplatz 2
II-5	Anschrift:	D-80333 München
		Deutschland
II-6	Staatsangehörigkeit (Staat)	DE
II-7	Sitz/Wohnsitz (Staat)	DE
II-8	Telefonnr.	(089) 636-82819
II-9	Telefaxnr.	(089) 636-81857

Handwritten signature/initials

PCT-ANTRAG

Original (für EINREICHUNG) - gedruckt am 13.03.2000 03:04:30 PM

III-1	Anmelder und/oder Erfinder	Anmelder und Erfinder Nur US LIGGESMEYER, Peter Am Langhölzl 18 D-85540 Haar Deutschland DE DE
III-1-1	Diese Person ist	
III-1-2	Anmelder für	
III-1-4	Name (FAMILIENNAME, Vorname)	
III-1-5	Anschrift:	
III-1-6	Staatsangehörigkeit (Staat)	
III-1-7	Sitz/Wohnsitz (Staat)	
IV-1	Anwalt oder gemeinsamer Vertreter; oder besondere Zustellanschrift Die unten bezeichnete Person ist/wird hiermit bestellt, um den (die) Anmelder vor den internationalen Behörden zu vertreten, und zwar als:	gemeinsamer Vertreter SIEMENS AKTIENGESELLSCHAFT Postfach 22 16 34 D-80506 München Deutschland (089) 636-82819 (089) 636-81857
IV-1-1	Name	
IV-1-2	Anschrift:	
IV-1-3	Telefonnr.	
IV-1-4	Telefaxnr.	
V	Bestimmung von Staaten	
V-1	Regionales Patent (andere Schutzrechtsarten oder Verfahren sind ggf. in Klammern nach der (den) betreffenden Bestimmung(en) angegeben)	EP: AT BE CH&LI CY DE DK ES FI FR GB GR IE IT LU MC NL PT SE und jeder weitere Staat, der Mitgliedsstaat des Europäischen Patentübereinkommens und Vertragsstaat des PCT ist
V-2	Nationales Patent (andere Schutzrechtsarten oder Verfahren sind ggf. in Klammern nach der (den) betreffenden Bestimmung(en) angegeben)	US
V-5	Erklärung bzgl. vorsorglicher Bestimmungen Zusätzlich zu den unter Punkten V-1, V-2 and V-3 vorgenommenen Bestimmungen nimmt der Anmelder nach Regel 4.9 Absatz b auch alle anderen nach dem PCT zulässigen Bestimmungen vor mit Ausnahme der nachstehend unter Punkt V-6 angegebenen Staaten. Der Anmelder erklärt, daß diese zusätzlichen Bestimmungen unter dem Vorbehalt einer Bestätigung stehen und jede zusätzliche Bestimmung, die vor Ablauf von 15 Monaten ab dem Prioritätsdatum nicht bestätigt wurde, nach Ablauf dieser Frist als vom Anmelder zurückgenommen gilt.	
V-6	Staaten, die von der Erklärung über vorsorgliche Bestimmungen ausgenommen werden	KEINE

PCT-ANTRAG

Original (für EINREICHUNG) - gedruckt am 13.03.2000 03:04:30 PM

VI-1	Priorität einer früheren nationalen Anmeldung beansprucht		
VI-1-1	Anmeldedatum	02 Juni 1999 (02.06.1999)	
VI-1-2	Aktenzeichen	19925239.4	
VI-1-3	Staat	DE	
VI-2	Ersuchen um Erstellung eines Prioritätsbeleges Das Anmeldeamt wird ersucht, eine beglaubigte Abschrift der in der (den) nachstehend genannten Zeile(n) bezeichneten früheren Anmeldung(en) zu erstellen und dem internationalen Büro zu übermitteln:	VI-1	
VII-1	Gewählte internationale Recherchenbehörde	Europäisches Patentamt (EPA) (ISA/EP)	
VIII	Kontrollliste	Anzahl der Blätter	Elektronische Datei(en) beigelegt
VIII-1	Antrag	4	-
VIII-2	Beschreibung	83	-
VIII-3	Ansprüche	5	-
VIII-4	Zusammenfassung	1	99_p_1974_p.txt
VIII-5	Zeichnung(en)	12	-
VIII-7	INSGESAMT	1045	-
	Beigefügte Unterlagen	Unterlage(n) in Papierform beigelegt	Elektronische Datei(en) beigelegt
VIII-8	Blatt für die Gebührenberechnung	✓	-
VIII-16	PCT-EASY-Diskette	-	Diskette
VIII-17	Sonstige (einzeln aufgeführt):	Kopie der Ursprungsfassung	-
VIII-18	Nr. der Abb. der Zeichn., die mit der Zusammenf. veröffentlicht werden soll	2	
VIII-19	Sprache der int. Anmeldung	Deutsch	
IX-1	Unterschrift des Anmelders oder Anwalts	i. V. Marg	
IX-1-1	Name	SIEMENS AKTIENGESELLSCHAFT	
IX-1-2	Name der unterzeichnenden Person	Margraf	
IX-1-3	Eigenschaft	Nr. 144/74 Ang.-AV	
IX-2	Unterschrift des Anmelders oder Anwalts	P. Liggesmeyer	
IX-2-1	Name (FAMILIENNAME, Vorname)	LIGGESMEYER, Peter	

Ode
28.04.00

VOM ANMELDEAMT AUSZUFÜLLEN

10-1	Datum des tatsächlichen Eingangs dieser internationalen Anmeldung	
10-2	Zeichnung(en):	
10-2-1	Eingegangen	
10-2-2	Nicht eingegangen	

PCT-ANTRAG

Original (für EINREICHUNG) - gedruckt am 13.03.2000 03:04:30 PM

10-3	Geändertes Eingangsdatum aufgrund nachträglich, jedoch fristgerecht eingeg. Unterlage(n) oder Zeichnung(en) zur Vervollständigung dieser int. Anmeldung	
10-4	Datum des fristgerechten Eingangs der Berichtigung nach PCT Artikel 11(2)	
10-5	Internationale Recherchenbehörde	ISA/EP
10-6	Übermittlung des Recherchenexemplars bis zur Zahlung der Recherchegebühr aufgeschoben	

VOM INTERNATIONALEN BÜRO AUSZUFÜLLEN

11-1	Datum des Eingangs des Aktenexemplars beim Internationalen Büro	
------	---	--

**PCT (ANHANG - BLATT FÜR DIE
GEBÜHRENBERECHNUNG)**

Original (für EINREICHUNG) - gedruckt am 13.03.2000 03:04:30 PM

(Dieses Blatt zählt nicht als Blatt der internationalen Anmeldung und ist nicht Teil derselben)

0	Vom Anmeldeamt auszufüllen			
0-1	Internationales Aktenzeichen.			
0-2	Eingangsstempel des Anmeldeamts			
0-4	Formular - PCT/RO/101 (Anlage)			
0-4-1	PCT Blatt für die Gebührenberechnung erstellt durch Benutzung von	PCT-EASY Version 2.90 (aktualisiert 08.03.2000)		
0-9	Aktenzeichen des Anmelders oder Anwalts	99P1974P		
2	Anmelder	SIEMENS AKTIENGESELLSCHAFT, et al.		
12	Berechnung der vorgeschriebenen Gebühren	Höhe der Gebühr/Multiplikator	Gesamtbeträge (DEM)	
12-1	Übermittlungsgebühr T	⇒	175	
12-2	Recherchegebühr S	⇒	1.848.26	
12-3	Internationale Gebühr Grundgebühr (erste 30 Blätter) b1	799.93		
12-4	Anzahl der Blätter über 30	74		
12-5	Zusatzblattgebühr (X)	17.6		
12-6	Gesamtbetrag der weiteren Gebühren b2	1.302.4		
12-7	b1 + b2 = B	2.102.33		
12-8	Bestimmungsgebühren Anzahl der in der internationalen Anmeldung vorgenommenen Bestimmungen	2		
12-9	Number of designation fees payable (maximum 8)	2		
12-10	Bestimmungsgebühr (X)	172.11		
12-11	Gesamtbetrag der Bestimmungsgebühren D	344.22		
12-12	PCT-EASY-Gebührenermäßigung R	-246.43		
12-13	Gesamtbetrag der internationalen Gebühr (B+D-R) I	⇒	2.200.12	
12-14	Gebühr für Prioritätsbeleg Anzahl der beantragten Prioritätsbelege	1		
12-15	Gebühr per Prioritätsbeleg (X)	35		
12-16	Gesamtbetrag Gebühr für Prioritätsbeleg(e) P	⇒	35	
12-17	Gesamtbetrag der zu zahlenden Gebühren (T+S+I+P)	⇒	4.258.38	
12-19	Zahlungsart	Sonstige: Abbuchung durch gesonderte Zahlungsliste		

**PCT (ANHANG - BLATT FÜR DIE
GEBÜHRENBERECHNUNG)**

Original (für EINREICHUNG) - gedruckt am 13.03.2000 03:04:30 PM

12-20	Anweisungen betreffend laufendes Konto Das Anmeldeamt:	Deutsches Patent- und Markenamt (RO/DE)
12-20-2	wird beauftragt, Fehlbeträge oder Überzahlungen des vorstehend angegebenen Gesamtbetrags der Gebühren meinem laufenden Konto zu belasten bzw. gutzuschreiben	✓
12-21	Nummer des laufenden Kontos	409022601
12-22	Datum	13 März 2000 (13.03.2000)
12-23	Name und Unterschrift	SIEMENS AKTIENGESELLSCHAFT <i>i. V. Murg</i>

PRÜFPROTOKOLL UND BEMERKUNGEN

13-2-1	Prüfergebnisse Antrag	Grün? Die Bezeichnung der Erfindung muß kurz und genau gefaßt sein. Bitte überprüfen.
13-2-2	Prüfergebnisse Staaten	Grün? Es können mehr Bestimmungen vorgenommen werden. Die folgenden Staaten sind nicht bestimmt worden: AP: (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW); EA: (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM); OA: (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG); AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, LI, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW. Bitte überprüfen.

Beschreibung

Verfahren und Anordnung zur Ermittlung einer Gesamtfehlerbe-
schreibung zumindest eines Teils eines Computerprogramms so-
5 wie Computerprogramm-Erzeugnis und computerlesbares Speicher-
medium

Die Erfindung betrifft ein Verfahren und eine Anordnung zur
Ermittlung einer Gesamtfehlerbeschreibung zumindest eines
10 Teils eines Computerprogramms sowie ein Computer-Erzeugnis
und ein computerlesbares Speichermedium.

Ein solches Verfahren und eine solche Anordnung ist aus [1]
bekannt.

15

Aus [1] ist bekannt, eine Gesamtfehlerbeschreibung in Form
eines Gesamtfehlerbaums für ein Computerprogramm rechnerge-
stützt zu ermitteln. Für das Computerprogramm wird eine Kon-
trollflußbeschreibung in Form eines Kontrollflußgraphen er-
20 mittelt. Für verschiedene Programmelemente des Computerpro-
gramms wird unter Verwendung einer gespeicherten Fehlerbe-
schreibung, die jeweils einem gespeicherten Referenzelement
zugeordnet ist, eine Elementenfehlerbeschreibung ermittelt.
Mit der Fehlerbeschreibung eines Referenzelements werden mög-
25 liche Fehler des jeweiligen Referenzelements beschrieben. Aus
den Elementenfehlerbeschreibungen in Form von Elementenfeh-
lerbäumen wird die Gesamtfehlerbeschreibung unter Berücksich-
tigung des Kontrollflußgraphen zu den Computerprogramm ermit-
telt.

30

Das Verfahren und die Anordnung aus [1] weisen insbesondere
folgende Nachteile auf. Der ermittelte Gesamtfehlerbaum ist
hinsichtlich der untersuchten Fehler und deren Ursachen un-
vollständig und damit unzuverlässig. Somit ist diese Vorge-
35 hensweise für sicherheitskritische Anwendungen im Rahmen der
Fehlerbaumgenerierung für ein Computerprogramm nicht sinnvoll
einsetzbar. Auch sind die einzelnen Fehlerbäume, die den Re-

ferenzelementen zugeordnet sind, unvollständig und damit unzuverlässig.

In [2] ist eine Übersicht über ein sogenanntes Slicing zu
5 finden. Slicing entspricht der Analyse, die bei der Ursachen-
suche für ein Fehlverhalten eines Computerprogrammes durchge-
führt wird. Im Rahmen dieses Vorgehens wird geprüft, ob das
Fehlverhalten durch eine aktuell betrachtete Anweisung verur-
sacht wurde. Ist dies nicht der Fall, so werden die Anweisun-
10 gen überprüft, die für die Anweisung Daten liefern oder ihre
Ausführung steuern. Dieses Verfahren wird fortgesetzt, bis
keine Vorgänge mehr existieren, also Eingabedaten des Compu-
terprogramms erreicht werden. Beim Slicing werden sogenannte
Slices ermittelt. Mit einem Slice wird dargestellt, welche
15 Anweisungen auf welche Weise von einem betrachteten Wert be-
einflußt werden. Im weiteren wird unter dem Begriff Slicing
stets ein rückwärts gerichtetes Slicing verstanden.

Aus [3] ist es bekannt, zu einem Computerprogramm eine Kon-
20 trollflußbeschreibung und eine Datenflußbeschreibung zu er-
mitteln. In [3] wird diese Darstellung als Ausgangsbasis für
sogenanntes datenflußorientiertes Testen des Computerpro-
gramms eingesetzt. Den Anweisungen (Knoten) des Kontrollfluß-
graphen werden Datenflußattribute (Datenflußbeschreibung) zu-
25 geordnet, die die Art der in den Anweisungen des Computerpro-
grammes enthaltenen Datenzugriffe beschreibt. Es werden
schreibende Zugriffe und lesende Zugriffe unterschieden.
Schreibzugriffe werden als Definitionen (def) bezeichnet. Le-
sende Zugriffe werden als Referenz bezeichnet. Erfolgt ein
30 lesender Zugriff in einer Entscheidung, so wird dieser Zu-
griff als prädikative Referenz (p-use, predicate use) be-
zeichnet. Ein lesender Zugriff in einer Berechnung eines Wer-
tes wird als berechnende Referenz bezeichnet (c-use, computa-
tional use).

35

Aus [4] sind Grundlagen über einen Fehlerbaum bekannt. Unter
einem Fehlerbaum ist, wie in [4] beschrieben, eine Struktur

zu verstehen, die logische Zusammenhänge zwischen Eingangsgrößen des Fehlerbaums beschreibt, welche Eingangsgrößen zu einem vorgegebenen unerwünschten Ereignis führen.

- 5 Ferner sind aus [5] verschiedene Verfahren zur Fehlerbaumanalyse bekannt.

Der Erfindung liegt das Problem zugrunde, eine Gesamtfehlerbeschreibung zu ermitteln, die gegenüber einer gemäß dem Verfahren aus [1] bekannten Ermittlung eines Gesamtfehlerbaums zuverlässiger ist.

Das Problem wird durch das Verfahren sowie durch die Anordnung mit den Merkmalen gemäß den unabhängigen Ansprüchen sowie durch das Computer-Erzeugnis und das computerlesbare Speichermedium mit den Merkmalen gemäß den unabhängigen Ansprüchen gelöst.

Bei einem Verfahren zur Ermittlung einer Gesamtfehlerbeschreibung zumindest eines Teils eines Computerprogrammes, durch einen Computer, ist zumindest der Teil des Computerprogrammes gespeichert. Es wird eine Kontrollflußbeschreibung und eine Datenflußbeschreibung zu dem Teil des Computerprogrammes ermittelt und es werden Programmelemente aus dem Teil des Computerprogrammes ausgewählt. Für jedes ausgewählte Programmelement wird unter Verwendung einer gespeicherten Fehlerbeschreibung eine Elementenfehlerbeschreibung ermittelt. Die Fehlerbeschreibung ist jeweils einem Referenzelement zugeordnet. Mit der Elementenfehlerbeschreibung werden mögliche Fehler des jeweiligen Programmelementes beschrieben. Mit einer Fehlerbeschreibung eines Referenzelements werden mögliche Fehler des jeweiligen Referenzelements beschrieben. Aus den Elementenfehlerbeschreibungen wird unter Berücksichtigung der Kontrollflußbeschreibung und der Datenflußbeschreibung die Gesamtfehlerbeschreibung ermittelt.

Eine Anordnung zur Ermittlung einer Gesamtfehlerbeschreibung zumindest eines Teils eines Computerprogrammes weist einen Prozessor auf, der derart eingerichtet ist, daß folgende Verfahrensschritte durchführbar sind:

- 5 - Zumindest der Teil des Computerprogramms ist gespeichert,
- es werden eine Kontrollflußbeschreibung und eine Datenflußbeschreibung für den Teil des Computerprogramms ermittelt,
- aus dem Teil des Computerprogrammes werden Programmelemente ausgewählt,
- 10 - für jedes ausgewählte Programmelement wird unter Verwendung einer gespeicherten Fehlerbeschreibung, die jeweils einem Referenzelement zugeordnet ist, eine Elementenfehlerbeschreibung ermittelt, mit der mögliche Fehler des jeweiligen Programmelements beschrieben werden,
- 15 - mit einer Fehlerbeschreibung eines Referenzelements werden mögliche Fehler des jeweiligen Referenzelements beschrieben,
- aus den Elementenfehlerbeschreibungen wird die Gesamtfehlerbeschreibung unter Berücksichtigung der Kontrollflußbeschreibung und der Datenflußbeschreibung ermittelt.
- 20

Ein Computerprogramm-Erzeugnis umfaßt ein computerlesbares Speichermedium, auf dem ein Programm gespeichert ist, daß es einem Computer ermöglicht, nachdem es in einen Speicher des Computers geladen worden ist, folgende Schritte durchzuführen zur Ermittlung einer Gesamtfehlerbeschreibung zumindest eines Teils eines Computerprogramms:

- Zumindest der Teil des Computerprogramms ist gespeichert,
- 30 - es werden eine Kontrollflußbeschreibung und eine Datenflußbeschreibung für den Teil des Computerprogramms ermittelt,
- aus dem Teil des Computerprogrammes werden Programmelemente ausgewählt,
- 35 - für jedes ausgewählte Programmelement wird unter Verwendung einer gespeicherten Fehlerbeschreibung, die jeweils einem Referenzelement zugeordnet ist, eine Elementenfehler-

- lerbeschreibung ermittelt, mit der mögliche Fehler des jeweiligen Programmelements beschrieben werden,
- mit einer Fehlerbeschreibung eines Referenzelements werden mögliche Fehler des jeweiligen Referenzelements beschrieben,
 - aus den Elementenfehlerbeschreibungen wird die Gesamtfehlerbeschreibung unter Berücksichtigung der Kontrollflußbeschreibung und der Datenflußbeschreibung ermittelt.
- 10 Auf einem computerlesbaren Speichermedium ist ein Programm gespeichert, daß es einem Computer ermöglicht, nachdem es in einen Speicher des Computers geladen worden ist, folgende Schritte durchzuführen zur Ermittlung einer Gesamtfehlerbeschreibung zumindest eines Teils eines Computerprogramms:
- Zumindest der Teil des Computerprogramms ist gespeichert,
 - es werden eine Kontrollflußbeschreibung und eine Datenflußbeschreibung für den Teil des Computerprogramms ermittelt,
 - aus dem Teil des Computerprogrammes werden Programmelemente ausgewählt,
 - für jedes ausgewählte Programmelement wird unter Verwendung einer gespeicherten Fehlerbeschreibung, die jeweils einem Referenzelement zugeordnet ist, eine Elementenfehlerbeschreibung ermittelt, mit der mögliche Fehler des jeweiligen Programmelements beschrieben werden,
 - mit einer Fehlerbeschreibung eines Referenzelements werden mögliche Fehler des jeweiligen Referenzelements beschrieben,
 - aus den Elementenfehlerbeschreibungen wird die Gesamtfehlerbeschreibung unter Berücksichtigung der Kontrollflußbeschreibung und der Datenflußbeschreibung ermittelt.

Durch die Erfindung ist es nunmehr möglich, eine zuverlässige, die Eigenheiten eines Computerprogramms berücksichtigende Gesamtfehlerbeschreibung für ein Computerprogramm oder einen Teil desselben zu ermitteln. Da die ermittelte Gesamtfehlerbeschreibung wesentlich zuverlässiger ist als die gemäß dem

Verfahren aus [1] ermittelbare Gesamtfehlerbeschreibung, ist die Erfindung auch für sicherheitskritische Anwendungen, d.h. insbesondere für die Ermittlung einer Gesamtfehlerbeschreibung eines sicherheitskritischen Computerprogramms geeignet.

5

Bevorzugte Weiterbildungen der Erfindung ergeben sich aus den abhängigen Ansprüchen.

Die Kontrollflußbeschreibung und/oder die Datenflußbeschreibung kann/können in Form eines Kontrollflußgraphen bzw. eines Datenflußgraphen vorliegen.

Die Fehlerbeschreibung kann in Form eines gespeicherten Fehlerbaums vorliegen und die Elementenfehlerbeschreibung kann als Elementenfehlerbaum ermittelt werden. In diesem Fall kann die Gesamtfehlerbeschreibung als Gesamtfehlerbaum ermittelt werden.

Durch diese Weiterbildung ist eine standardisierte Darstellung einer Fehlerbeschreibung möglich, was es einem Benutzer der Fehlerbeschreibung erheblich vereinfacht, diese zu analysieren.

Die Gesamtfehlerbeschreibung kann in einer Weiterbildung eingesetzt werden zur Fehleranalyse des Teils des Computerprogramms.

Diese Weiterbildung weist insbesondere den Vorteil auf, daß eine automatisierte, zuverlässige Fehleranalyse, bei Vorliegen der Fehlerbeschreibungen in Form von Fehlerbäumen sogar eine gemäß den Fehlerbaumanalyseverfahren "normierte" Analyse der Fehlerbeschreibung möglich wird.

In einer weiteren Ausgestaltung wird die Gesamtfehlerbeschreibung als Gesamtfehlerbaum ermittelt und der Gesamtfehlerbaum wird hinsichtlich vorgegebbarer Rahmenbedingungen verändert.

Die Veränderung kann durch Hinzufügen eines Ergänzungsfehlerbaums erfolgen.

- 5 Ein Ausführungsbeispiel der Erfindung ist in den Figuren dargestellt und wird im weiteren näher erläutert:

Es zeigen

- 10 Figur 1 einen Computer, mit dem das Verfahren gemäß dem Ausführungsbeispiel durchgeführt wird;
Figur 2 ein Ablaufdiagramm, in dem die einzelnen Verfahrensschritte des Verfahrens gemäß dem Ausführungsbeispiel dargestellt sind;
- 15 Figur 3 eine Darstellung eines allgemeinen Fehlerbaums, wie er für ein Referenzelement prinzipiell gebildet wird;
Figuren 4a bis 4c einen Kontrollflußgraphen (Figur 4a), einen Slice (Figur 4b) und einen Fehlerbaum (Figur 4c) für eine Anweisungssequenz als Referenzelement eines Computerprogramms;
- 20 Figuren 5a bis 5c einen Kontrollflußgraphen (Figur 5a), einen Slice (Figur 5b) und einen Fehlerbaum (Figur 5c) für eine Auswahlsequenz als Referenzelement eines Computerprogramms;
- 25 Figur 6a bis 6c einen Kontrollflußgraphen (Figur 6a), einen Slice (Figur 6b) und einen Fehlerbaum (Figur 6c) für eine Schleife als Referenzelement eines Computerprogramms;
- Figur 7 ein Kontrollflußgraph mit Datenflußgraph zu einem
30 Computerprogramm gemäß dem Ausführungsbeispiel;
Figuren 8a und 8b einen Slice der Ausgabe der Variable max (Figur 8a) bzw. einen Slice zu der Variable avr (Figur 8b) zu dem Programm gemäß dem Ausführungsbeispiel;
- 35 Figur 9 den Slice für die Variable avr, in dem eine Struktur der Schleife aus dem Programm des Ausführungsbeispiels hervorgehoben ist;

Figur 10 einen Fehlerbaum für die Annahme, daß die Variable avr fehlerhaft ist;

Figur 11 den Gesamtfehlerbaum gemäß Figur 10, wobei redundante Ereignisse aus dem Gesamtfehlerbaum gemäß Figur 10 zu einem Ereignis zusammengefaßt worden sind.

Fig.1 zeigt einen Computer 100 mit dem das im weiteren beschriebene Verfahren durchgeführt wird.

10 Der Computer 100 weist einen Prozessor 101 auf, der über einen Bus 103 mit einem Speicher 102 verbunden ist. Mit dem Bus 103 ist ferner eine Eingangs-/Ausgangsschnittstelle 106 verbunden.

15 In dem Speicher 102 ist ein Computerprogramm 104 gespeichert, für das auf die im folgenden beschriebene Weise eine Gesamtfehlerbeschreibung ermittelt wird. Ferner ist in dem Speicher 102 ein Programm 105 gespeichert, durch das das im weiteren beschriebene Verfahren realisiert ist. Ferner sind
20 in dem Speicher Fehlerbeschreibungen 115 unterschiedlicher Referenzelemente eines Computerprogramms gespeichert. Mit einer Fehlerbeschreibung eines Referenzelements werden mögliche Fehler des jeweiligen Referenzelements beschrieben. Verschiedene Referenzelemente und den Referenzelementen zugeordnete
25 Fehlerbeschreibungen werden im weiteren detailliert erläutert.

Mit der Eingangs-/Ausgangsschnittstelle 106 ist über eine erste Verbindung 107 eine Tastatur 108 verbunden. Über eine
30 zweite Verbindung 109 ist die Eingangs-/Ausgangsschnittstelle 106 mit einer Computermouse 110 und über eine dritte Verbindung 111 ist die Eingangs-/Ausgangsschnittstelle 106 mit einem Bildschirm 112 verbunden, auf dem die ermittelte Gesamtfehlerbeschreibung des Computerprogramms 104 dargestellt
35 wird. Über eine vierte Verbindung 113 ist die Eingangs-/Ausgangsschnittstelle 106 mit einem externen Speichermedium 114 verbunden.

Fig.2 zeigt in einem Blockschaltbild die Vorgehensweise gemäß dem im weiteren beschriebenen Ausführungsbeispiel.

- 5 Aus dem gespeicherten Computerprogramm 104 werden ein Kontrollflußgraph 201 und ein Datenflußgraph 202 für das Computerprogramm 104 ermittelt.

10 Aus dem Computerprogramm werden einzelne Programmelemente ausgewählt (Schritt 203). Für jedes ausgewählte Programmelement wird unter Verwendung einer gespeicherten Fehlerbeschreibung, die einen zu dem ausgewählten Programmelement korrespondierenden Referenzelement zugeordnet ist, eine Elementenfehlerbeschreibung ermittelt (Schritt 204). Mit der
15 Elementenfehlerbeschreibung werden mögliche Fehler des jeweiligen ausgewählten Programmelements beschrieben.

Ausgehend von einem von einem Benutzer vorgegebenen zu untersuchenden Fehlerereignis in dem Computerprogramm (unerwünschtes Ereignis) wird in einem letzten Schritt (Schritt 205) eine Gesamtfehlerbeschreibung des Computerprogrammes für den zu untersuchenden Fehlerfall ermittelt aus den Elementenfehlerbeschreibungen, wobei der Kontrollflußgraph und der Datenflußgraph berücksichtigt werden.

25 Der ermittelte Gesamtfehlerbaum wird dem Benutzer auf dem Bildschirm 112 dargestellt.

Fig.3 zeigt die grundlegende Vorgehensweise bei der Erstellung eines Fehlerbaums, wie sie im Rahmen des Ausgangsbeispiels verwendet worden ist zur Bildung der im weiteren beschriebenen Fehlerbäume zu den Referenzelementen.

30 Zu einem von einem Benutzer ausgewählten Ereignis 301 ist zu ermitteln, wie das ausgewählte fehlerhafte Ereignis entstehen kann. Bei einem Computerprogramm kann eine fehlerhafte Ausgabe einer Variablen als ausgewähltes fehlerhaftes Ereignis
35

(unerwünschtes Ereignis) 301 verursacht werden durch einen Kontrollflußfehler 303 und/oder einen Datenfehler 304 (INKLUSIV-ODER-Verknüpfung 302).

- 5 Unter einem Kontrollflußfehler 303 ist eine fehlerhafte Ansteuerung der Verarbeitung der jeweiligen Variablen zu verstehen.

Unter dem Datenflußfehler 304 ist ein Fehler zu verstehen,
10 der durch fehlerhafte Daten bei der Verarbeitung entsteht.

Der Datenflußfehler 304 kann in dem aktuell betrachteten Verarbeitungsschritt neu entstehen (Block 306) und/oder er kann schon vorhanden gewesen sein und lediglich durch Fehlerpropagation erhalten bleiben (Block 307) (INKLUSIV-ODER-Verknüpfung
15 305).

Ausgehend von diesen Überlegungen werden im weiteren für folgende Elemente eines Computerprogrammes jeweils der entsprechende Fehlerbaum, ein die Anweisung beschreibender Slice sowie ein Kontrollflußgraph dargelegt:

- eine Anweisungssequenz,
- ein Auswahllement,
- ein Schleifenelement.

25

Anweisungssequenz

Die Anweisungssequenz 401 weist die in **Fig.4a** dargestellten drei Anweisungen auf. In einer ersten Anweisung 402 wird einer ersten Variable j der Wert 3 zugewiesen ($j := 3$). Durch eine zweite Anweisung 403 wird einer zweiten Variable k der Wert 2 zugeordnet ($k := 2$). Durch eine dritte Anweisung 404 wird eine Summe über die erste Variable und die zweite Variable gebildet ($i := j + k$).

30

35

Es wird gemäß der Vorgehensweise aus [2] zu dieser Anweisungssequenz 401 ein Slice 410 gebildet, wie er in **Fig.4b** dargestellt ist. Die erste Anweisung 402 und die zweite Anweisung 403 haben beide Auswirkungen auf die dritte Anweisung 404, was durch zwei Pfeile 411, 412 in dem Slice 410 dargestellt ist.

Zu dem Kontrollflußgraph 401 ergibt sich der in **Fig.4c** dargestellte Fehlerbaum 420 für folgendes vorgegebenes unerwünschtes Ereignis 421:

"Variable i ist nach der dritten Anweisung fehlerhaft".

Das fehlerhafte Ereignis 421 kann durch einen Fehler bei der betrachteten dritten Anweisung 404 selbst bei bis zu diesem Anweisungsschritt korrekten Daten erzeugt worden sein (Element 422 in **Fig.4c**). Das fehlerhafte Ereignis 421 kann jedoch auch durch verfälschte Eingabedaten der dritten Anweisung verursacht werden, d.h. durch INKLUSIV-ODER-Verknüpfung 424 der Ereignisse, daß die zweite Variable k nach der zweiten Anweisung fehlerhaft war (Element 425) und/oder daß die erste Variable j nach der ersten Anweisung 402 fehlerhaft war (Element 426). Das Ergebnis der ersten INKLUSIV-ODER-Verknüpfung 424 wird inklusiv-oder verknüpft mit dem Ereignis, daß die dritte Anweisung fehlerhaft ist (INKLUSIV-ODER-Verknüpfung 423).

Auswahlelement

30

Bei einem Auswahlelement als Referenzelement müssen Fehlermöglichkeiten der Datenflüsse und der Kontrollflüsse innerhalb des Computerprogramms beachtet werden.

35 In den **Fig.5a** bis **Fig.5c** ist ein Kontrollflußgraph 501 (vgl. **Fig.5a**), ein Slice 520 (vgl. **Fig.5b**) sowie ein Fehlerbaum 540

(vgl. **Fig.5c**) für eine If-Then-Else-Anweisung als Auswahl-
element dargestellt.

Der Kontrollflußgraph 501 umfaßt folgende sechs Anweisungen:

- 5 - Eine erste Anweisung 502, mit der einer ersten Variable j
der Wert 3 zugeordnet wird ($j := 3$),
- eine zweite Anweisung 503, mit der einer zweiten Variable
 k ein vorgebbarer Wert zugeordnet wird ($k := \dots$),
- 10 - eine dritte Anweisung 504, in der überprüft wird, ob die
zweite Variable k einen Wert größer als 0 aufweist; ist
der Wert der zweiten Variable größer 0, dann verzweigt die
Anweisung zu einer vierten Anweisung 505, sonst zu einer
fünften Anweisung 506,
- die vierte Anweisung 505, in der einer dritten Variable i
15 der Wert der zweiten Variable k zugeordnet wird ($i := k$),
- eine fünfte Anweisung 506, in der der dritten Variable i
der Wert der zweiten Variable k mit negativem Vorzeichen
zugeordnet wird ($i := -k$),
- 20 - eine sechste Anweisung 507, in der die dritte Variable i
in beliebiger Weise weiterverarbeitet wird.

Zu dem in **Fig.5a** dargestellten Kontrollflußgraphen 501 ergibt
sich für das Auswahl-
element der in **Fig.5b** dargestellte Slice
520.

25

Durchgezogene Kanten in dem Slice 520 stellen eine Datenab-
hängigkeit der unterschiedlichen Anweisungen voneinander dar.

30 Mit gestrichelten Kanten werden Kontrollabhängigkeiten der
entsprechenden Anweisungen voneinander angegeben.

Für die beiden Kantentypen gelten die folgenden Definitionen:

- gestrichelte Kanten, im weiteren als Kontrollkanten be-
zeichnet, sind von Anweisungen, die eine prädikative Refe-
35 renz enthalten (Ausfallkonstrukte, Schleifensteuerung),
auf die unmittelbar kontrollierten Anweisungen gerichtet,
d.h. auf jene Anweisungen, die nur ausgeführt werden, wenn

- das Prädikat einen bestimmten Wert hat. Kontrollkanten werden nur zwischen der kontrollierenden Anweisung und unmittelbar eingeschachtelten Anweisungen gezogen. Ist in einem kontrollierten Block eine weitere Kontrollebene eingeschachtelt, so werden keine Kontrollkanten gezogen, die mehr als eine Ebene überstreichen. Da eine Kontrollbeziehung transitiv ist, kann diese mittelbare Kontrolle aus dem Slice durch Ausnutzung der Transitivität geschlossen werden.
- 5
- 10 - durchgezogene Kanten, im weiteren als Datenflußkanten bezeichnet, sind von Anweisungen, in denen eine Variable definiert wird, auf Anweisungen gerichtet, in denen diese Variable referenziert wird. Die betrachtete Variable darf zwischen der Definition und der Referenz nicht erneut definiert werden. Man bezeichnet dies als definitionsfreien
- 15 Pfad bezüglich der betrachteten Variablen.

Der Slice wird ermittelt, indem ausgehend von der Anweisung mit der betrachteten Variablen, für die das unerwünschte Ereignis vorgegeben wird, der Kontrollflußgraph gegen die Kantenrichtung nach Definition der betrachteten Variablen durchsucht wird. Existieren zu der Definition berechnende Referenzen, so wird das Verfahren rekursiv fortgesetzt, bis keine zusätzlichen Knoten mehr gefunden werden. Die auf diese Weise gefundenen Abhängigkeiten zwischen Anweisungen sind Datenabhängigkeiten. Befindet sich ein betrachteter Knoten in einem Block, dessen Ausführung von einer Entscheidung direkt gesteuert wird, so stellt dies eine Kontrollabhängigkeit dar. Für die prädikativen Referenzen der in der Entscheidung beteiligten Variablen werden Knoten mit entsprechenden Definitionen - also Datenflußabhängigkeiten - rekursiv gesucht, die weitere Kontrollabhängigkeiten besitzen.

20

25

30

Fig.5b zeigt den zu dem Ausfallelement gehörenden Slice 520 mit entsprechenden Kontrollkanten und Datenflußkanten.

35

Fig.5c zeigt den Fehlerbaum 540 für das vorgegebene Ereignis "die dritte Variable i ist vor der 6. Anweisung fehlerhaft" 541.

- 5 Folgende Ereignisse führen in INKLUSIV-ODER-Verknüpfung 542 zu dem fehlerhaften Ereignis 541:
- eine UND-Verknüpfung 543 der Ereignisse, daß die Entscheidung gemäß der dritten Anweisung 504 wahr ist (Element 544) und einem Ergebnis einer INKLUSIV-ODER-Verknüpfung 545 der Ereignisse, daß die vierte Anweisung 505 fehlerhaft ist (Element 546) und/oder die erste Variable j nach der ersten Anweisung 502 fehlerhaft ist (Element 547);
 - eine UND-Verknüpfung 550 der Ereignisse, daß die Entscheidung gemäß der dritten Anweisung 504 falsch ist (Element 551) mit einem Ergebnis einer INKLUSIV-ODER-Verknüpfung 552 der Ereignisse, daß die fünfte Anweisung fehlerhaft ist (Element 553) und/oder daß die erste Variable j nach der ersten Anweisung 502 fehlerhaft ist (Element 554);
 - eine INKLUSIV-ODER-Verknüpfung 560 der Ereignisse: Die Entscheidung gemäß der dritten Anweisung 504 ist fehlerhaft (Element 561) und/oder die zweite Variable k ist nach der zweiten Anweisung 503 fehlerhaft (Element 562).

25 **Mehrfach-Auswahlelement**

Ein Mehrfach-Auswahlelement als Referenzelement kann nach dem oben beschriebenen Schema durch Aufbrechen der Mehrfachauswahl in eine Kaskade von zweiseitigen Auswahlelementen, die gemäß dem oberen Vorgehen bearbeitet werden, behandelt werden, um somit einen Fehlerbaum für ein Mehrfach-Auswahlelement zu ermitteln.

35 **Schleife**

Die **Fig.6a** bis **Fig.6c** zeigen für das Referenzelement einer Schleife einen Fehlerbaum 601 (vgl. **Fig.6a**), den entsprechenden Slice 620 (vgl. **Fig.6b**) sowie den zugehörigen Fehlerbaum 640 (vgl. **Fig.6c**).

5

Der Kontrollflußgraph 601 für ein Schleifenelement weist folgende sieben Anweisungen auf:

- Eine erste Anweisung 602, mit der einer ersten Variable i der Wert 0 zugeordnet wird ($i := 0$),
- 10 - eine zweite Anweisung 603, mit der einer zweiten Variable j ein Wert frei zugeordnet wird ($j := \dots$),
- eine dritte Anweisung 604, durch die einer dritten Variable k ein weiterer Wert frei vorgegeben wird ($k := \dots$),
- 15 - eine vierte Anweisung 605, die als Schleifenanweisung eine Bedingung angibt, daß eine fünfte Anweisung sowie eine sechste Anweisung solange durchgeführt werden, solange der Wert der zweiten Variablen $j > 0$ ist (WHILE $j > 0$ DO),
- eine fünfte Anweisung 606, in der der ersten Variable i ein Wert zugeordnet wird, der sich ergibt aus der Summe
- 20 des bisherigen Werts der ersten Variable sowie dem Produkt aus der zweiten Variable und der dritten Variable ($i := i + k * j$),
- eine sechste Anweisung 607, durch die der zweiten Variable j ein Wert zugewiesen wird, der sich ergibt durch Verminderung des ursprünglichen Werts der zweiten Variable j um
- 25 den Wert 1 ($j := j - 1$),
- eine siebte Anweisung 608, in der die erste Variable i in vorgegebbarer Weise weiterbearbeitet wird ($\dots := i \dots$).

30 **Fig.6b** zeigt den entsprechenden Slice 620 zu dem in **Fig.6a** dargestellten Kontrollflußgraphen 601 mit zugehörigen Kontrollflußkanten und Datenflußkanten.

35 Der in **Fig.6c** dargestellte Fehlerbaum 640 wird gebildet für das vorgegebene Ereignis 641, daß die "erste Variable i vor der siebten Anweisung fehlerhaft" ist.

Der Fehlerbaum 640 ergibt sich durch INKLUSIV-ODER-Verknüpfung 642 folgender vier Ereignisse:

- Ein erstes Ereignis 643, das beschreibt, daß die erste Variable i nach der ersten Anweisung 602 fehlerhaft ist,
- 5 - eine UND-Verknüpfung 644 aus den Ereignissen, daß der Schleifenrumpf der Schleife mindestens zweimal durchlaufen worden ist (Element 645) und dem Ereignis, daß die sechste Anweisung 607 fehlerhaft ist (646),
- eine UND-Verknüpfung 650 des Ereignisses, daß der Schleifenrumpf mindestens einmal ausgeführt wurde (Element 651) und einer INKLUSIV-ODER-Verknüpfung 652 folgende vier Ereignisse:
 - a) die fünfte Anweisung 606 ist fehlerhaft (Element 653),
 - b) die erste Variable i ist nach der ersten Anweisung fehlerhaft (Element 654)
 - 15 c) die zweite Variable j ist nach der zweiten Anweisung fehlerhaft (Element 655),
 - d) die dritte Variable k ist nach der dritten Anweisung fehlerhaft (Element 656),
- 20 - eine INKLUSIV-ODER-Verknüpfung 660 folgender drei Ereignisse:
 - e) die Entscheidung gemäß der vierten Anweisung 605 ist fehlerhaft (Element 661),
 - f) die zweite Variable j ist nach der zweiten Anweisung 603 fehlerhaft (Element 662),
 - 25 g) eine UND-Verknüpfung 663 der Ereignisse, daß die sechste Anweisung fehlerhaft ist (Element 664) mit dem Ereignis, daß der Schleifenrumpf mindestens einmal durchlaufen worden ist (Element 665).

30

Die oben beschriebenen Fehlerbäume, die den einzelnen Referenzelementen zugeordnet sind, sind in dem Speicher 102 als Fehlerbäume 115 gespeichert.

Fig.7 zeigt einen Kontrollflußgraphen 700 zu folgendem Computerprogramm:

```

input (n);
5  input (a);
   max:=0;
   sum:=0;
   i:=2;
   WHILE i =n DO
10      IF max < a[i]
         THEN max:= a[i]
            sum:= sum + a[i]
            i:= i + 1
   avr:= sum/n;
15  output (max);
   output (avr);

```

Zu dem in **Fig.7** dargestellten Kontrollflußgraphen 700 mit 13 Anweisungen (Bezugszeichen 1, 2, 3, ..., 13) zeigt **Fig.8a** den zugehörigen Slice 800 zu der Variablen **max** und **Fig.8b** den zugehörigen Slice 810 zu der Variablen **avr**. Die Numerierung der einzelnen Anweisungen in den Slices entspricht der Numerierung der einzelnen Anweisungen in dem Kontrollflußgraphen 700 aus **Fig.7**.

Fig.9 zeigt den Slice 900 für die Variable **avr**, wie er in **Fig.8b** dargestellt ist. Die Struktur des in dem oben dargestellten Programm enthaltenen Schleifenelements ist durch Fettdruck hervorgehoben. Diese Struktur entspricht dem in **Fig.6b** dargestellten Slice für ein Schleifenelement.

Ein Gesamtfehlerbaum 1000 für das oben dargestellte Computerprogramm ist in **Fig.10** dargestellt. Der Gesamtfehlerbaum für das Computerprogramm wird durch Instantiieren des entsprechenden Fehlerbaums, der dem Referenzelement zugeordnet ist, das dem ausgewählten Programmelement entspricht, erzeugt.

Durch rückwärts gerichtetes Vorgehen ausgehend von dem vorgegebenen unerwünschten Ereignis wird somit unter Verwendung der den Referenzelementen zugeordneten Fehlerbäume der Gesamtfehlerbaum 1000 ermittelt.

5

In Fig.10 ist der Fehlerbaum 1000 zu dem Ereignis, daß "die Variable **avr** vor der dreizehnten Anweisung fehlerhaft" ist (Element 1001). Die Variable **avr** kann vor der dreizehnten Anweisung 13 fehlerhaft sein aufgrund mindestens eines der folgenden drei Ereignisse, wie dies auch in dem in Fig.9 dargestellten Slice 900 für die Variable **avr** dargestellt ist (INKLUSIV-ODER-Verknüpfung 1002):

- Eine Eingangsvariable **n** ist nach der ersten Anweisung 1 fehlerhaft (Element 1003),
- 15 - die elfte Anweisung 11 ist fehlerhaft (Element 1004),
- der Wert der Variablen **sum** vor der elften Anweisung 11 ist fehlerhaft (Element 1005).

Die Variable **sum** ist vor der elften Anweisung 11 fehlerhaft (Element 1005), wenn mindestens eines der folgenden Ereignisse erfüllt ist (INKLUSIV-ODER-Verknüpfung 1006):

- Die Variable **sum** ist nach der vierten Anweisung 4 fehlerhaft (Element 1007),
- eine UND-Verknüpfung 1008 des Ereignisses, daß mindestens zweimal der Schleifenrumpf ausgeführt worden ist (Element 25 1009) mit dem Ereignis, daß die zehnte Anweisung 10 fehlerhaft ist (Element 1010),
- eine UND-Verknüpfung 1011 des Ereignisses, daß mindestens einmal der Schleifenrumpf ausgeführt worden ist (Element 30 1012) mit dem Ergebnis einer INKLUSIV-ODER-Verknüpfung 1013 folgender vier Ereignisse:
 - a) die neunte Anweisung 9 ist fehlerhaft (Element 1014),
 - b) die Variable **sum** ist nach der vierten Anweisung 4 fehlerhaft (Element 1015),
 - 35 c) die Variable **i** ist nach der fünften Anweisung 5 fehlerhaft (Element 1016),

- d) die Variable a ist nach der zweiten Anweisung 2 fehlerhaft (Element 1017),
 - eine INKLUSIV-ODER-Verknüpfung 1018 folgender Ereignisse:
 - e) die Entscheidung gemäß der sechsten Anweisung ist fehlerhaft (Element 1019),
 - f) die Variable i ist nach der fünften Anweisung fehlerhaft (Element 1020),
 - g) die Variable n ist nach der ersten Anweisung fehlerhaft (Element 1021),
 - h) eine UND-Verknüpfung 1022 des Ereignisses, daß die 10. Anweisung fehlerhaft ist (Element 1023) mit dem Ereignis, daß mindestens einmal der Schleifenrumpf ausgeführt worden ist (Element 1024).
- Der Fehlerbaum 1000 aus **Fig.10** wird zur anschaulicheren Darstellung dahingehend verändert, daß Ereignisse, die in dem Fehlerbaum 1000 mehrfach dargestellt sind, zu einem Knoten eines Ursache-Wirkungs-Graphen 1100 (vgl. **Fig.11**) zusammengefaßt werden.
- Auf den in **Fig.10** dargestellten Fehlerbaum 1000 wird ein Fehlerbaumanalyseverfahren, wie in [5] beschrieben, angewendet, wodurch eine Analyse des Computerprogramms hinsichtlich eines vorgegebenen unerwünschten Ereignisses analysiert wird.
- Im weiteren werden Alternativen und weitere Anwendungsmöglichkeiten des oben beschriebenen Ausführungsbeispiels dargestellt.
- Der mit dem oben beschriebenen Verfahren erzeugte Gesamtfehlerbaum kann zu verschiedenen Zwecken eingesetzt werden:
- Beschreibung der Fehlererzeugung bzw. Fehlverhaltenspropagation durch einen Teil eines Computerprogramms im Rahmen einer Sicherheitsanalyse oder eine Zuverlässigkeitsanalyse des Computerprogramms,
 - Analyse von Softwarefehlermechanismen, beispielsweise im Rahmen einer Testfallgenerierung.

Im weiteren ist ein Computerprogramm in der Programmiersprache C++ angegeben, mit dem das Verfahren gemäß dem Ausführungsbeispiel realisiert ist:

```

5      #include "AS_GraphKante.h"
        □
        #include <iostream>
        □
10     □
        ostream& operator << ( ostream& os, const GraphKanteC & Kante){
        □
            os << Kante.KantenTyp;
15     □
            return os;
        □
        }

20     //////////////////////////////////////
        // Klasse zur Erzeugung von Kanten im Slice-Graph.
        // Es gibt zwei Arten von Kanten:
        //      1. Kontrollfluß-Kanten KFK
        //      2. Datenfluß-Kanten DFK
25     // Auch diese Klasse erfüllt die nice-Anforderungen für die STL.
        // //////////////////////////////////////
        // 1997-09-12 Andreas Steinhorst -
        // //////////////////////////////////////
30     #ifndef _GraphNodeHeader
        #define _GraphNodeHeader

        #include <iostream>
        #include "KFGLListNode.h"
35     using namespace std;

        class GraphNodeC : public KFGLListNodeC {
        public:
40         //GraphNodeC()
                                {};

        friend ostream& operator << ( ostream& os, const GraphNodeC& Node);
45     };

        #endif

50     //////////////////////////////////////
        □
        // Klasse zur Erstellung eines Slice. Dieser Klasse wird ein leeres Objekt
        □
        // der Klasse Graph vererbt.
55     □
        // Die Ecken des Graphen/Slice haben dieselbe Struktur wie die Listen im KFG.
        // Die Kanten sind von Typ KantenTypT; eine Klasse GraphKanteC ließ sich aus
        // für mich unersichtlichen Gründen nicht in die Klasse Graph einbinden.
        //
60     // //////////////////////////////////////
        // 1997-09-12 Andreas Steinhorst -
        // //////////////////////////////////////
        #ifdef _SliceHeader
        #else
65     #define _SliceHeader

        #include "graph.h"
        #include "AS_GraphKante.h"
        // #include "AS_GraphNode.h"
70     #include "KFGLListNode.h"
        #include "KFGLList.h"

        //using namespace std;
        //typedef enum {DFK, KFK} KantenTypT;

```

```

class SliceC : public Graph<KFGListNodeC, GraphKanteC> {
public:
5   SliceC() {};

   void sliceForLoops(KFGListC & L2, KFGListC::iterator LOOPIT);
10  KFGListC::iterator defineVariableToSlice(KFGListC & L2);
   //void buildTestGraph(KFGListC & L2);

   void findAllDefs(KFGListC & L1, KFGListC::iterator ITER);
15  KFGListC::iterator findUpperLimit(KFGListC & L1, KFGListC::iterator ITER,
KFGLP_UseListC::iterator PUSEIT);

   KFGListC::iterator findLowerLimit(KFGListC & L1, KFGListC::iterator ITER,
20  KFGListC::iterator UPPERLIMIT);

   KFGListC::iterator findLowerLimitFromCUse(KFGListC & L1, KFGListC::iterator ITER,
KFGListC::iterator UPPERLIMIT);

25  KFGListC::iterator findUpperLimitFromCUse(KFGListC & L1, KFGListC::iterator ITER,
KFGLUseListC::iterator USEIT);

   int checkForNodes(int NodeNumber);
30  void KFKPUseToCUse(KFGListC::iterator PUSEIT, int SchleifenEnde);
   void findDefToCUse(KFGListC & L1, KFGListC::iterator ITER);

   void defInLoop(KFGListC & L1, KFGListC::iterator ITER);
35  void startBuildSlice(KFGListC & L1);
   void sliceAusgeben();

40  };

#ifdef
45  ////////////////////////////////////////
// Mit einigen Änderungen übernommen aus "Die C++ Standard Template Library".
//
// Template-Klasse Graph zur Erstellung von gerichteten oder ungerichteten
// Graphen. Der Graph besteht aus einem Vektor E für alle Ecken. Jedes
50  // Vektorelement besteht wiederum aus einem Paar: der Ecke und der Menge der
// Nachfolger.
// Die Menge der Nachfolger wird durch den STL Datentyp map dargestellt; der
// Schlüssel zu einem Kantentyp/Kantenwert ist die Nummer einer nachfolgenden
// Ecke.
55  //
// ////////////////////////////////////////
#ifdef _graphHeader
#define _graphHeader
#include <assert.h>
60  #include <map>
#include <stack>
#include <vector>
#include "checkvec.h"
#include <iostream>
65  #include "AS_GraphNode.h"
using namespace std;

// Leere Parameterklasse mit Mindestsatz von Operationen,
// falls keine Kantengewichte benötigt werden.
70  struct Empty
{
    public:
        Empty(int=0) {}
        bool operator<(const Empty&) const { return true;}
75  };
// Empty-Operationen, damit Ein-Ausgabe allgemein formuliert
// werden kann
// ostream& operator<<(ostream& os, const Empty&) { return os;}

```

```

//istream& operator>>(istream& is, Empty& ) { return is;}

5  template<class Eckentyp, class Kantentyp>
   class Graph
   {
       public:
           typedef map<int,Kantentyp, less<int> > Nachfolger;
           typedef pair<Eckentyp, Nachfolger> Ecke;
           typedef checkedVector<Ecke> Graphtyp;
           //typedef vector<Ecke> Graphtyp;
           typedef Graphtyp::iterator iterator;
           typedef Graphtyp::const_iterator const_iterator;

15     Graph(bool g, ostream& os = cerr)
        : gerichtet(g), pOut(&os) {}

           Graph() {}

20     size_t size() const { return C.size(); }
        bool istGerichtet() const { return gerichtet;}
        iterator begin() { return C.begin();}
        iterator end() { return C.end();}
        Ecke& operator[](int i) { return C[i];}

25     size_t AnzahlKanten();
        int insert(const Eckentyp& e);
        void insert(const Eckentyp& e1, const Eckentyp& e2,
           const Kantentyp& Wert);
30     void verbindeEcken(int e1, int e2, // über Eckennummern
           const Kantentyp& Wert);
        void setgerichtet (bool wert); // neue Methode für gerichtete/ungerichtete Gra-
phen,
// die aber nicht ver-
35     wendet wird.
        void check(ostream& = cout);
        void ZyklusUndZusammenhang(ostream& = cout);

40     private:
        bool gerichtet;
        Graphtyp C; // Container
        ostream* pOut;
    };

45     //Funktion, die überprüft, ob es sich um einen gerichteten oder ungerichteten
    //Graphen handelt, und die Anzahl der Ecken und Kanten ausgibt.
    template<class Eckentyp, class Kantentyp>
    void Graph<Eckentyp,Kantentyp>::check(ostream& os)
50     {
        os << "Der Graph ist ";
        //if(!istGerichtet())
        // os << "un";
        os << "gerichtet und hat "
55         << size() << " Knoten und "
        << AnzahlKanten() << " Kanten\n";
        //ZyklusUndZusammenhang(os);
    }

60     //Methode, die einen Wert setzt, ob der Graph gerichtet oder ungerichtet ist.
    template<class Eckentyp, class Kantentyp>
    void Graph<Eckentyp,Kantentyp>::setgerichtet(bool wert)
65     {
        gerichtet = wert;
    }

70     //Funktion berechnet die Anzahl der Kanten in Graph
    template<class Eckentyp, class Kantentyp>
    size_t Graph<Eckentyp,Kantentyp>::AnzahlKanten()
    {
75         size_t Kanten = 0;
        iterator temp = begin();
        while(temp != end())
            Kanten += (*temp++).second.size();
        //if(!gerichtet)
        // Kanten /= 2;
    }

```

```

    return Kanten;
}

5 //Einfügen einer Ecke in den Graphen, falls diese noch nicht vorhanden ist.
  template<class Eckentyp, class Kantentyp>
  int Graph<Eckentyp, Kantentyp>::insert(const Eckentyp& e)
  {
10      for(int i = 0; i < size(); ++i)
          if(e == C[i].first)
              return i;

          // falls nicht gefunden, einfüegen:
15      C.push_back(Ecke(e, Nachfolger()));
      return size()-1;
  }

20 //Einfügen einer Kante in den Graphen, indem zuerst die Ecken eingefügt werden.
  template<class Eckentyp, class Kantentyp>
  void Graph<Eckentyp, Kantentyp>::insert(const Eckentyp& e1,
                                          const Eckentyp& e2,
                                          const Kantentyp& Wert)
  {
25      int pos1 = insert(e1);
          int pos2 = insert(e2);
          verbindeEcken(pos1, pos2, Wert);
  }

30 //Verbinden der beiden neu eingefügten Ecken durch eine Kante.
  template<class Eckentyp, class Kantentyp>
  void Graph<Eckentyp, Kantentyp>::verbindeEcken(
35      int pos1, int pos2, const Kantentyp& Wert)
  {
      (C[pos1].second)[pos2] = Wert;
      //if(!gerichtet) // automatisch auch Gegenrichtung eintragen
      // (C[pos2].second)[pos1] = Wert;
40  }

/* ZyklusUndZusammenhang() arbeitet mit der Tiefensuche.
   Im Gegensatz zu CLR S. 478 wurde nicht mit der Rekursion
   gearbeitet, weil bei dieser Stack-Overflow bei großen Graphen
45   (zB. MILES.DAT bei mehr als 40 Knoten) auftrat.
   Die Nachbildung der Rekursion durch eigenen Stack
   ermöglicht die Verarbeitung der gesamten Datei
   MILES.DAT (128 Knoten). Die Stacktiefe entspricht der Kantenanzahl
   + 1 bei ungerichteten Graphen.
50 */

  template<class Eckentyp, class Kantentyp>
  void Graph<Eckentyp, Kantentyp>::ZyklusUndZusammenhang(ostream& os)
  {
55      int Zyklen = 0;
          int Komponentenanzahl = 0;
          stack<int, vector<int> > > EckenStack; // zu besuchende Ecken

          // allen Ecken den Zustand nichtBesucht zuordnen
60      enum EckStatus {nichtBesucht, besucht, bearbeitet};
          vector<EckStatus> Eckenzustand(size(), nichtBesucht);

          // alle Ecken besuchen
          for(int i = 0; i < size(); ++i)
65              if(Eckenzustand[i] == nichtBesucht)
                  {
                      Komponentenanzahl++;
                      // auf dem Stack zwecks Bearbeitung ablegen
70                      EckenStack.push(i);

                      // Stack abarbeiten
                      while(!EckenStack.empty())
                      {
75                          int dieEcke = EckenStack.top();
                              EckenStack.pop();
                              if(Eckenzustand[dieEcke] == besucht)
                                  Eckenzustand[dieEcke] = bearbeitet;
                              else

```



```

    if(Eckenzustand[dieEcke] == nichtBesucht)
    {
        Eckenzustand[dieEcke] = besucht;
        // neue Ecke, für bearbeitet-Kennung vormerken
        EckenStack.push(dieEcke);

        // Nachfolger vormerken:
        Graph<Eckentyp, Kantentyp>::Nachfolger::iterator
        start = operator[] (dieEcke).second.begin(),
        ende = operator[] (dieEcke).second.end();
        while(start != ende)
        {
            int Nachf = (*start).first;
            if(Eckenzustand[Nachf] == besucht)
            {
                ++Zyklen; // hier war schon jemand!
                (*pOut) << "mindestens Ecke "
                << operator[] (Nachf).first
                << " ist in einem Zyklus\n";
            }
            if(Eckenzustand[Nachf] == nichtBesucht)
                EckenStack.push(Nachf);
            ++start;
        }
    } // Stack Empty?
} // for() ... if(Eckenzustand...

if(gerichtet)
{ if(Komponentenanzahl == 1)
  os << "Der Graph ist stark zusammenh.,ngend.\n";
  else
    os << "Der Graph ist nicht oder schwach "
    "zusammenh.,ngend.\n";
}
else
  os << "Der Graph hat "
  << Komponentenanzahl
  << " Komponente(n)." << endl;

os << "Der Graph hat ";
if(Zyklen == 0)
  os << "keine ";
os << "Zyklen." << endl;
}

//Ausgabe des Graphen.
template<class Eckentyp, class Kantentyp>
ostream& operator<<(ostream& os,
    Graph<Eckentyp,Kantentyp>& G)
{
    ofstream Ziel("FaultTree.uwg");
    ostream_iterator<uwgknotenC> POSIT(Ziel);
    ostream_iterator<uwgknotenC> POSIT2(Ziel);
    // Anzeige der Ecken mit Nachfolgern
    Ziel << "%%UWG" << endl << "\V0.1\"~\"Fehlerbaum\"~\"A.Steinhorst\" " << endl <<
    "%%BEGIN" << endl;
    for(int i = 0; i < G.size(); ++i)
    {
        POSIT++ = G[i].first;
        /*os << G[i].first << " <";
        Graph<Eckentyp,Kantentyp>::Nachfolger::iterator
        startN = G[i].second.begin(),
        endeN = G[i].second.end();
        while(startN != endeN)
        {
            os << G[(startN).first].first << ' ' // Ecke
            << (startN).second << ' '; // Kantenwert
            *POSIT2++ = G[(startN).first].first;
            /*KANTEIT++ = (startN).second;
            ++startN;
        }
        os << ">\n";*/
        *POSIT++;
    }

    for(int u = 0; u < G.size(); ++u) {
        Graph<Eckentyp,Kantentyp>::Nachfolger::iterator

```

25

```

startN = G[u].second.begin(),
endeN = G[u].second.end();
while (startN != endeN) {
    Ziel << "%%EDGE:" << G[u].first.getGatterId() << ","
    << G[*startN].first.first.getGatterId() << "/0;" << endl;
    ++startN;
}
Ziel << "%%PROBSLIST" << endl << "%%END" << endl;
return os;
}

#endif

#ifdef _KFGDefHeader
□
#else
□
#define _KFGDefHeader
□

□
#include <string>
□
#include <iostream>
□

□
using namespace std;
□

□
typedef char StringT [256];

□
class KFGDefC {
□
private:
□

StringT Def;
int ScopeLevelD;

public:
KFGDefC() { strcpy (Def, "-- unknown --");
ScopeLevelD = 0; };

KFGDefC(char _Def [], int _ScopeLevelD) { strcpy (Def, _Def);
ScopeLevelD =
_ScopeLevelD; };

void setDef(char _Def []) { strcpy (Def, _Def); };
char* getDef() { return Def; };
int getScopeLevelD() { return ScopeLevelD; };
bool operator == (const KFGDefC& other) const { return ((strcmp(Def,
other.Def) == 0) & (Scope-
LevelD == other.ScopeLevelD)); };

bool operator != (const KFGDefC& other) const { return !(*this ==
other); };

bool operator < (const KFGDefC& other) const { return (strcmp(Def,
other.Def) < 0); };

bool operator > (const KFGDefC& other) const

```

```

other.Def) > 0); };

friend ostream& operator << ( ostream& os, const KFGLDefC& Node);

5      );

#endif

10  //////////////////////////////////////
    D
    // Klasse für Hilfsliste, um daraus einen KFG zu erstellen.
    // Diese Klasse erfüllt die nice-Anforderungen für die STL;
    // d.h. für eine Klasse T gilt
15  // 0. Sie unterstützt den Copy-Konstruktor T (const T&),
    // 1. den Zuweisungsoperator T& operator= (const T&),
    // 2. den Vergleichsoperator bool operator== (const T, &const T&) und
    // 3. den Ungleichoperator !=
    // in einer Weise so daß gilt:
20  // (a) { TRUE } T a(b) { a == b }
    // (b) { TRUE } a = b { a == b }
    // (c) { a == a }
    // (d) { a == b <==> b == a }
    // (e) { (a == b) AND ( b == c ) ==> (a == c) }
25  // (f) { a != b <==> NOT ( a == b ) };
    //
    // Außerdem sind alle Funktionen, insbesondere getName equality preserving, d.h.
    // (g) { a == b ==> a.getName() == b.getName() }
    //
30  // (C) 1997 Siemens AG
    //
    // //////////////////////////////////////
    // 1997-08-25 Andreas Steinhorst -
    // //////////////////////////////////////
35  #ifdef _KFGLLineNodeHeader
    #else
    #define _KFGLLineNodeHeader

    #include <string>
40  #include <iostream>

    using namespace std;

    typedef char StringL [1000];
45
    class KFGLLineNodeC {
    private:

        StringL Name;
        int LineNumber;

    public:

        KFGLLineNodeC()
55  { strcpy (Name, "-- unknown --
        LineNumber = 0;

        KFGLLineNodeC(char _Name []) { strcpy (Name, _Name);
60  LineNumber =
        0;};

        KFGLLineNodeC(char _Name [], int _LineNumber)
65  { strcpy (Name,
        _Name);
        LineNumber =
        _LineNumber; };

70  void setName(char _Name []) { strcpy (Name, _Name); };

        char* getName() { return Name; };

        int getLineNumber() { return LineNumber; };
75  bool operator == (const KFGLLineNodeC& other) const
        { return
        ((strcmp(Name, other.Name) == 0)

```

```

other.LineNumber)); };                                     & (LineNumber ==

5         bool operator != (const KFGLineNodeC& other) const      { return !(*this
== other);    };

        bool operator < (const KFGLineNodeC& other) const          { return
10 (strcmp(Name, other.Name) < 0); };

        bool operator > (const KFGLineNodeC& other) const          { return
15 (strcmp(Name, other.Name) > 0); };

        friend ostream& operator << ( ostream& os, const KFGLineNodeC& Node);
    };

20 #endif

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Klasse für die Liste, die den KFG darstellt.
// Diese Klasse erfüllt die nice-Anforderungen für die STL;
25 // d.h. für eine Klasse T gilt
// 0. Sie unterstützt den Copy-Konstruktor T (const T&),
// 1. den Zuweisungsoperator T& operator= (const T&),
// 2. den Vergleichsoperator bool operator== (const T, &const T&) und
// 3. den Ungleichoperator !=
30 // in einer Weise so daß gilt:
// (a) { TRUE } T a(b) { a == b }
// (b) { TRUE } a = b { a == b }
// (c) { a == a }
// (d) { a == b <==> b == a }
35 // (e) { (a == b) AND ( b == c ) ==> (a == c) }
// (f) { a != b <==> NOT ( a== b ) };
//
// Außerdem sind alle Funktionen, insbesondere getStatement equality preserving, d.h.
40 // (g) { a == b ==> a.getStatement() == b.getStatement() }
//
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// 1997-09-01 Andreas Steinhorst -
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
45 #ifdef _KFGListHeader
#else
#define _KFGListHeader

50 #include <string>
#include <iostream>
#include <stdio.h>
#include <list>
#include <fstream>
#include <iterator>
55 #include "KFGLListNode.h"
#include "KFGTokenList.h"
#include "KFGProgList.h"

60 using namespace std;

class KFGListC : public list<KFGLListNodeC> {

65     int Anzahl_Defs, Anzahl_Uses, Anzahl_P_Uses, Anzahl_Deklarationen;
    int Schleifenentscheidungen, Entscheidungen, Zaehlschleifenentscheidungen;
    int Anweisungen;

public:
70     KFGListC() {};

    void TokenList2KFGList(KFGTokenListC & L1);

    void KnotenNummern();

75     void KnotenIdentifizierer(KFGListC::iterator KFG);

    void ListeAusgeben();

```

```

void addLineInToList();

void addLineToKFG(KFGProgListC & LP);

5   int zaehleDeklarationen(KFGTokenListC & TL);

void basisgroessenInDatei();

10  };

    class KFGDefListC : public list<KFGDefC> {};

    class KFGUseListC : public list<KFGUseC> {};

15  class KFGP_UseListC : public list<KFGUseC> {};

    #endif

20  #include <string>
    □
    #include <iostream>
    #include <list>
25  #include "KFGNode.h"

    using namespace std;

30  typedef list<KFGNodeC> KFGListeT;

    extern void KFGListeAusgeben(KFGListeT);

35  //////////////////////////////////////
    □
    // Klasse für die Knoten im Kontrollflußgraph.
    // Diese Klasse erfüllt die nice-Anforderungen für die STL;
    // d.h. für eine Klasse T gilt
40  // 0. Sie unterstützt den Copy-Konstruktor T (const T&),
    // 1. den Zuweisungsoperator T& operator= (const T&),
    // 2. den Vergleichsoperator bool operator== (const T, &const T&) und
    // 3. den Ungleichoperator !=
    // in einer Weise so daß gilt:
45  // (a) { TRUE } T a(b) { a == b }
    // (b) { TRUE } a = b { a == b }
    // (c) { a == a }
    // (d) { a == b <==> b == a }
    // (e) { (a == b) AND ( b == c ) ==> (a == c) }
50  // (f) { a != b <==> NOT ( a == b ) };
    //
    // Außerdem sind alle Funktionen, insbesondere getStatement equality preserving, d.h.
    // (g) { a == b ==> a.getStatement() == b.getStatement() }
55  //
    // 1997-08-25 Andreas Steinhorst
    //////////////////////////////////////
    #ifdef _KFGNodeListHeader
    #else
60  #define _KFGNodeListHeader

    #include <string>
    #include <iostream>
    #include <list>
65  #include "KFGUse.h"
    #include "KFGDef.h"

    using namespace std;

    typedef char StringT [256];
    typedef char CodeLineT[256];
    typedef enum
75  {NO,LC,BL,EL,IFC,BTh,ETh,BEL,EEL,OP,DOWL,DOWLC,SWITCH,CASE,ENDCASE,BDEFA,ENDDEFA,RETURN,BRE
    AK} KFGNodeTypeT;
    typedef enum {LOOP, THEN, ELSE, NONE} KnotenIdentT;

    class KFGListNodeC {

```

```

protected:
//private:

5         static int DummyNodeNr;
          int NodeNr;
          int Level;
          int KnotenNr;
          int LineNr;
10        StringT Statement;
          CodeLineT CodeLine;
          KFGNodeTypeT KFGNodeType;
          KnotenIdentT KnotenIdent ;

15        public:

          list<KFGUseC> Uses;
          list<KFGDefC> Defs;
          list<KFGUseC> P_Uses;

20        KFGListNodeC()                                { strcpy (Statement, "-- unknown
--");                                                    Level = 0;
                                                         KFGNodeType =
                                                         NodeNr = Dum-
                                                         KnotenNr = 0;
                                                         KnotenIdent =
30        NONE;                                           LineNr = 0;
                                                         strcpy (CodeLi-
ne, "--unknown--"); };

35        KFGListNodeC(char _Statement [], KFGNodeTypeT _KFGNodeType) { strcpy (State-
ment, _Statement);                                     Level = 0;
                                                         KFGNodeType =
                                                         NodeNr = Dum-
                                                         KnotenNr = 0;
                                                         KnotenIdent =
40        _KFGNodeType;                                   LineNr = 0;
                                                         strcpy (CodeLi-
myNodeNr++;                                           ne, "--unknown--"); };

45        NONE;                                           KFGListNodeC(char _Statement [], KFGNodeTypeT _KFGNodeType, int _Level)
                                                         { strcpy (State-
ment, _Statement);                                     Level = _Level;
                                                         KFGNodeType =
                                                         NodeNr = Dum-
                                                         KnotenNr = 0;
                                                         KnotenIdent =
50        KFGListNodeC(char _Statement [], KFGNodeTypeT _KFGNodeType, int _Level)
                                                         { strcpy (State-
ment, _Statement);                                     Level = _Level;
                                                         KFGNodeType =
                                                         NodeNr = Dum-
                                                         KnotenNr = 0;
                                                         KnotenIdent =
55        _KFGNodeType;                                   LineNr = 0;
                                                         strcpy (CodeLi-
myNodeNr++;                                           ne, "--unknown--"); };

60        NONE;                                           KFGListNodeC(char _Statement [], KFGNodeTypeT _KFGNodeType, int _Level, int
_LineNr)
                                                         { strcpy (State-
ment, _Statement);                                     Level = _Level;
                                                         KFGNodeType =
                                                         NodeNr = Dum-
                                                         KnotenNr = 0;
                                                         KnotenIdent =
65        KFGListNodeC(char _Statement [], KFGNodeTypeT _KFGNodeType, int _Level, int
_LineNr)
                                                         { strcpy (State-
ment, _Statement);                                     Level = _Level;
                                                         KFGNodeType =
                                                         NodeNr = Dum-
                                                         KnotenNr = 0;
                                                         KnotenIdent =
70        _KFGNodeType;                                   LineNr =
                                                         LineNr =
75        NONE;
          _LineNr;

```

```

ne, "--unknown--"); };
                                strcpy (CodeLi-

5         void setStatement(char _Statement [])
                                { strcpy (State-
ment, _Statement); };

        void setCodeLine(char _CodeLine [])
                                { strcpy (CodeLi-
10 ne, _CodeLine); };

        void setKFGKnotenNummer(int _KnotenNr)
                                { KnotenNr =
15 _KnotenNr; };

        void setKnotenIdent (KnotenIdentT _KnotenIdent)
                                { KnotenIdent =
_KnotenIdent; };

20         int getKnotenNummer()      { return KnotenNr; };

        char* getStatement()          { return Statement; };

25         char* getCodeLine()         { return CodeLine; };

        int getLevel()                { return Level; };

        int getNodeNr()               { return NodeNr; };

30         int getLineNr()             { return LineNr; };

        KnotenIdentT getKnotenIdent() { return KnotenIdent; };

35         KFGNodeTypeT getKFGNodeType() { return KFGNodeType; };

        bool operator == (const KFGLListNodeC& other) const
                                { return
40 ((strcmp(Statement, other.Statement) == 0)
                                & (Level ==
                                & (KFGNodeType
                                & (NodeNr ==
50 other.NodeNr) )); };

        bool operator != (const KFGLListNodeC& other) const
                                { return !(*this
== other); };

55         bool operator < (const KFGLListNodeC& other) const
                                { return
(strcmp(Statement, other.Statement) < 0); };

        bool operator > (const KFGLListNodeC& other) const
                                { return
60 (strcmp(Statement, other.Statement) > 0); };

        friend ostream& operator << ( ostream& os, const KFGLListNodeC& Node);

#ifdef
65 // #include <string>
// #include <iostream>
// #include "KFGLList.h"

// using namespace std;

70 // class KFGLListOpC : public KFGLListeC {
// public:

75 //     void KFGLListOut(KFGLListeT & L)
//     {
//         KFGLListeT::iterator I = L.begin();
//         while(I != L.end())

```

31

```

//          cout << *I++ << ' ';
//          cout << " size() = " << L.size() << endl;
//      }
//  };
5
///////////////////////////////////////////////////////////////////
10 // Klasse für Hilfsliste, um daraus einen KFG zu erstellen.
//
// Diese Klasse erfüllt die nice-Anforderungen für die STL;
// d.h. für eine Klasse T gilt
// 0. Sie unterstützt den Copy-Konstruktor T (const T&),
// 1. den Zuweisungsoperator T& operator= (const T&),
15 // 2. den Vergleichsoperator bool operator== (const T, &const T&) und
// 3. den Ungleichoperator !=
// in einer Weise so daß gilt:
// (a) { TRUE } T a(b) { a == b }
// (b) { TRUE } a = b { a == b }
20 // (c) { a == a }
// (d) { a == b <==> b == a }
// (e) { (a == b) AND ( b == c ) ==> (a == c) }
// (f) { a != b <==> NOT ( a== b ) };
//
25 // Außerdem sind alle Funktionen, insbesondere getName equality preserving, d.h.
// (g) { a == b ==> a.getName() == b.getName() }
//
// (C) 1997 Siemens AG
//
30 ///////////////////////////////////////////////////////////////////
// 1997-08-25 Andreas Steinhorst -
///////////////////////////////////////////////////////////////////
#ifdef _KFGNodeHeader
35 #else
#define _KFGNodeHeader

#include <string>
#include <iostream>

40 using namespace std;

typedef char StringT [256];

45 class KFGNodeC {
private:

    StringT Name;
    int ScopeLevel;

50 public:

    KFGNodeC() { strcpy (Name, "-- un-
known --"); ScopeLevel = 0; }

55 erzeugt" << endl ; */};

    KFGNodeC(char _Name [] ) { strcpy (Name, _Name);
                                ScopeLevel = 0; };

60 KFGNodeC(char _Name [], int _ScopeLevel)
    { strcpy (Name,
    _Name);
    ScopeLevel =
65 _ScopeLevel;};

    void setName(char _Name []) { strcpy (Name, _Name); };

    char* getName() { return Name; };

70 int getScopeLevel() { return ScopeLevel; };

    bool operator == (const KFGNodeC& other) const
    { return
75 ((strcmp(Name, other.Name) == 0)
    & (ScopeLevel
    == other.ScopeLevel)); };

    bool operator != (const KFGNodeC& other) const

```



```

== other);    };                                { return !(*this

5          bool operator < (const KFGNodeC& other) const      { return (strcmp(Name,
other.Name) < 0); };

          bool operator > (const KFGNodeC& other) const      { return (strcmp(Name,
10 other.Name) > 0); };

          friend ostream& operator << ( ostream& os, const KFGNodeC& Node);
    };
15 #endif

#ifdef _KFGNodeHeader
20 #else
    ///////////////////////////////////////////////////////////////////
    // Klasse für Hilfsliste, um daraus einen KFG zu erstellen.
    // Diese Klasse erfüllt die nice-Anforderungen für die STL;
25 // d.h. für eine Klasse T gilt
    // 0. Sie unterstützt den Copy-Konstruktor T (const T&),
    // 1. den Zuweisungsoperator T& operator= (const T&),
    // 2. den Vergleichsoperator bool operator== (const T, &const T&) und
    // 3. den Ungleichoperator !=
30 // in einer Weise so daß gilt:
    // (a) { TRUE } T a(b) {a == b}
    // (b) { TRUE } a = b {a == b}
    // (c) { a == a}
    // (d) { a == b <==> b == a }
35 // (e) { (a == b) AND ( b == c ) ==> (a == c) }
    // (f) { a != b <==> NOT ( a== b ) };
    //
    // Außerdem sind alle Funktionen, insbesondere getName equality preserving, d.h.
40 // (g) { a == b ==> a.getName() == b.getName() }
    //
    // (C) 1997 Siemens AG
    //
    ///////////////////////////////////////////////////////////////////
45 // 1997-08-25 Andreas Steinhorst
    ///////////////////////////////////////////////////////////////////
    #ifdef _KFGNodeHeader
    #else
    #define _KFGNodeHeader

50 #include <string>
    #include <iostream>

    using namespace std;

55     typedef char StringT [256];

    class KFGListNodeC {
    private:

60         StringT Name;
        int ScopeLevel;
        list <KFGUseC> Uses;
        list <KFGDefC> Defs;

65     public:

        KFGListNodeC()
known --"); ScopeLevel = 0;                                { strcpy (Name, "-- un-

70 erzeugt" << endl ; */);                                /* cout << "NodeC

        KFGListNodeC(char _Name [] )                    { strcpy (Name, _Name);
                                                         ScopeLevel = 0; };

75         KFGListNodeC(char _Name [], int _ScopeLevel)
        _Name);                                { strcpy (Name,

```

```

ScopeLevel =
_ScopeLevel;};

5      void setName(char _Name [])      { strcpy (Name, _Name);      };
      char* getName()                  { return Name; };
      int getScopeLevel()              { return ScopeLevel; };
10     bool operator == (const KFGLListNodeC& other) const
      ((strcmp(Name, other.Name) == 0)      { return
      == other.ScopeLevel)); };                  & (ScopeLevel

15     bool operator != (const KFGLListNodeC& other) const      { return !(*this
      == other);      };

20     bool operator < (const KFGLListNodeC& other) const      { return (strcmp(Name,
      other.Name) < 0); };

25     bool operator > (const KFGLListNodeC& other) const      { return (strcmp(Name,
      other.Name) > 0); };

      friend ostream& operator << ( ostream& os, const KFGLListNodeC& Node);
30     };
    #endif

35     #ifdef _KFGNodeHeader
    □
    #else
    □
    //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    □
40     // Klasse für Hilfsliste, um daraus einen KFG zu erstellen.
    □
    // Diese Klasse erfüllt die nice-Anforderungen für die STL;
    // d.h. für eine Klasse T gilt
    // 0. Sie unterstützt den Copy-Konstruktor T (const T&),
    // 1. den Zuweisungsoperator T& operator= (const T&),
45     // 2. den Vergleichsoperator bool operator== (const T, &const T&) und
    // 3. den Ungleichoperator !=
    // in einer Weise so daß gilt:
    // (a) { TRUE } T a(b) {a == b}
    // (b) { TRUE } a = b {a == b}
50     // (c) { a == a}
    // (d) { a == b <==> b == a }
    // (e) { (a == b) AND ( b == c ) ==> (a == c) }
    // (f) { a != b <==> NOT ( a== b) };
55     //
    // Außerdem sind alle Funktionen, insbesondere getName equality preserving, d.h.
    // (g) { a == b ==> a.getName() == b.getName() }
    //
    // (C) 1997 Siemens AG
60     //
    //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    // 1997-08-25 Andreas Steinhorst
    //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    #ifdef _KFGNodeHeader
65     #else
    #define _KFGNodeHeader

    #include <string>
    #include <iostream>
70     using namespace std;

    typedef char StringT [256];

75     class KFGLListNodeC {
    private:

        StringT Name;

```

```

        int ScopeLevel;
        list <KFGUseC> Uses;
        list <KFGDefC> Defs;

5      public:

        KFGListNodeC() { strcpy (Name, "-- un-
known --"); ScopeLevel = 0; }
10     erzeugt" << endl ; */};

        KFGListNodeC(char _Name [] ) { strcpy (Name, _Name);
                                         ScopeLevel = 0; };

15     KFGListNodeC(char _Name [], int _ScopeLevel)
        { strcpy (Name,
        _Name);
        ScopeLevel =
20     _ScopeLevel;};

        void setName(char _Name []) { strcpy (Name, _Name); };

        char* getName() { return Name; };

25     int getScopeLevel() { return ScopeLevel; };

        bool operator == (const KFGListNodeC& other) const
        { return
30     ((strcmp(Name, other.Name) == 0)
        & (ScopeLevel
        == other.ScopeLevel)); };

        bool operator != (const KFGListNodeC& other) const
35     { return !(*this
        == other); };

        bool operator < (const KFGListNodeC& other) const
        { return (strcmp(Name,
40     other.Name) < 0); };

        bool operator > (const KFGListNodeC& other) const
        { return (strcmp(Name,
45     other.Name) > 0); };

        friend ostream& operator << ( ostream& os, const KFGListNodeC& Node);

    #endif

50     //Klasse für eine Liste aus STL. KFGListeC erbt dabei alle Eigenschaften
    //von der mit STL erstellten Liste list<KFGNodeC>.
    #ifdef _KFGProgListHeader
    #else
55     #define _KFGProgListHeader

        #include <string>
        #include <iostream>
60     #include <stdio.h>
        #include <list>
        #include <fstream>
        #include <iterator>
65     #include "KFGLineNode.h"

        using namespace std;

70     class KFGProgListC : public list<KFGLineNodeC> {
    public:

        KFGProgListC() {};

75     //void KFGListeAusgeben();

        //void KFGTokenListToKFGList ();

```

```

};

#endif

5 //Klasse für eine Liste aus STL. KFGListeC erbt dabei alle Eigenschaften
  □
  //von der mit STL erstellten Liste list<KFGNodeC>.
  □
10 #ifndef _KFGTokenListHeader
  □
  #else
  □
15 #define _KFGTokenListHeader
  □

  □

20 #include <string>
  □
  #include <iostream>
  □
25 #include <stdio.h>
  #include <list>
  #include <fstream>
  #include <iterator>
  #include "KFGTokenNode.h"

30 using namespace std;

class KFGTokenListC : public list<KFGTokenNodeC> {
35 public:
    KFGTokenListC() {};

    void KFGListeAusgeben();

40    //void KFGTokenListToKFGList ();

};

45 #endif

////////////////////////////////////
50 □
  // Klasse für Hilfsliste, um daraus einen KFG zu erstellen.
  □
  // Diese Klasse erfüllt die nice-Anforderungen für die STL;
  □
  // d.h. für eine Klasse T gilt
55 // 0. Sie unterstützt den Copy-Konstruktor T (const T&),
  // 1. den Zuweisungsoperator T& operator= (const T&),
  // 2. den Vergleichsoperator bool operator== (const T, &const T&) und
  // 3. den Ungleichoperator !=
  // in einer Weise so daß gilt:
60 // (a) { TRUE } T a(b) {a == b}
  // (b) { TRUE } a = b {a == b}
  // (c) { a == a}
  // (d) { a == b <==> b == a }
  // (e) { (a == b) AND ( b == c ) ==> (a == c) }
65 // (f) { a != b <==> NOT ( a== b) };
  //
  // Außerdem sind alle Funktionen, insbesondere getName equality preserving, d.h.
  // (g) { a == b ==> a.getName() == b.getName() }
  //
70 // (C) 1997 Siemens AG
  //
  // //////////////////////////////////////
  // 1997-08-25 Andreas Steinhorst -
75 // //////////////////////////////////////
  #ifndef _KFGTokenNodeHeader
  #else
  #define _KFGTokenNodeHeader

```

```

#include <string>
#include <iostream>

using namespace std;

5     typedef char StringT [256];
        typedef enum
        {N, PL, IF, BT, ET, BE, EE, FOR, BF, EF, WHILE, BW, EW, SW, CA, ECA, BRK, RET, DEFA, EDEFA, DO, BDOW, DOWS, NL, DEF
10     ,BFUNC,EFUNC,TD,OUT,FOR_D,PF} TokenNodeTypeT;

        class KFGTokenNodeC {
        private:

15            StringT Name;
            TokenNodeTypeT TokenNodeType;
            int ScopeLevel;
            int LineNumber;

        public:

20            KFGTokenNodeC() { strcpy (Name, "-- un-
known --");
                                ScopeLevel = 0;
                                TokenNodeType =
25            N;
                                LineNumber = 0;
        };

            KFGTokenNodeC(char _Name [], TokenNodeTypeT _TokenNodeType) { strcpy (Name,
30            _Name);
                                ScopeLevel = 0;
                                TokenNodeType =
35            _TokenNodeType;
                                LineNumber =
0;};

            KFGTokenNodeC(char _Name [], int _ScopeLevel, TokenNodeTypeT _TokenNodeType)
40            { strcpy (Name,
                                ScopeLevel =
                _ScopeLevel;
                                TokenNodeType =
45            _TokenNodeType;
                                LineNumber = 0;
        };

            KFGTokenNodeC(char _Name [], int _ScopeLevel, TokenNodeTypeT _TokenNodeType,
50            int _LineNumber) { strcpy (Name,
                                ScopeLevel =
                _ScopeLevel;
                                TokenNodeType =
55            _TokenNodeType;
                                LineNumber =
                _LineNumber; };

60            void setName(char _Name []) { strcpy (Name, _Name); };
            char* getName() { return Name; };

65            int getScopeLevel() { return ScopeLevel; };
            int getLineNumber() { return LineNumber; };
            TokenNodeTypeT getTokenNodeType() { return TokenNodeType; };

70            bool operator == (const KFGTokenNodeC& other) const { return
((strcmp(Name, other.Name) == 0)
                                & (ScopeLevel
75            == other.ScopeLevel)
                                & (TokenNode-
Type == other.TokenNodeType)); };

            bool operator != (const KFGTokenNodeC& other) const

```

```

== other);    };                                { return !(*this

5          bool operator < (const KFGTokenNodeC& other) const      { return
  (strcmp(Name, other.Name) < 0); };

          bool operator > (const KFGTokenNodeC& other) const      { return
10  (strcmp(Name, other.Name) > 0); };

          friend ostream& operator << ( ostream& os, const KFGTokenNodeC& Node);
        };
15  #endif

        #ifdef _KFGUseHeader
        □
20      #else
        □
        #define _KFGUseHeader
        □

25      □
        #include <string>
        □
        #include <iostream>
        □
30      □
        using namespace std;
        □

35      □
        typedef char StringT [256];
        □

40      □
        class KFGUseC {
        □
        private:
        □

45      □
        StringT Use;
        □
        int ScopeLevelU;
        □
50      □
        public:
        □

55      □
        KFGUseC()                                { strcpy (Use, "-- unknown --");
                                                ScopeLevelU = 0; };

60      □
        KFGUseC(char _Use [], int _ScopeLevelU)
                                                { strcpy (Use, _Use);
                                                ScopeLevelU =
        _ScopeLevelU; };

65      □
        void setUse(char _Use [])    { strcpy (Use, _Use); };
        char* getUse()                { return Use; };
        int getScopeLevelU()        { return ScopeLevelU; };

70      □
        bool operator == (const KFGUseC& other) const
other.Use) == 0)                                { return ((strcmp(Use,
LevelU == other.ScopeLevelU)); };                & (Scope-

75      □
        bool operator != (const KFGUseC& other) const
other); };                                { return !(*this ==

```

```

        bool operator < (const KFGUseC& other) const      { return (strcmp(Use,
5      other.Use) < 0); };

        bool operator > (const KFGUseC& other) const      { return (strcmp(Use,
        other.Use) > 0); };

10      friend ostream& operator << ( ostream& os, const KFGUseC& Node);
        };

15      #endif

      #ifndef _uwggraphC
      □
20      #define _uwggraphC
      □

      □
      #include <Graph.h>
      □
25      #include "uwgknoten.h"
      □
      #include "uwgkante.h"
      □
30      #include "AS_Slice.h"

      using namespace std;

      class uwggraphC : public Graph < uwgknotenC, uwgkanteC > {
35      public:
          short int eingefaltet;

          uwggraphC() {};

40          SliceC::iterator findOutputNode(SliceC & S1);

          SliceC::iterator defineIterator(SliceC & S1, int SliceNr);

          void addFirstNodesToFT(SliceC & S1, int KnotenNr, int Pos1);
45          SliceC::iterator returnCondNode(SliceC & S1, int SliceNr);

          int checkUWGNode(int SliceNr);

50          void SliceAusgeben(SliceC & S1);

          void startBuildFT(SliceC & S1);

          void buildInLoopTree(SliceC & S1, int Pos1, SliceC::iterator SLPUSE, Sli-
55      ceC::iterator SLNODE);

          void buildIn_D1_Part(SliceC & S1, int posOR_D1, SliceC::iterator SLPUSE,
          SliceC::iterator SLNODE, int D2_RefNr);

60          SliceC::iterator findOutmostPUse(SliceC & S1, SliceC::iterator SLPUSE);

          int checkPUses1(SliceC & S1, SliceC::iterator SLPUSE, SliceC::iterator
          SLPUSEREF);

65          int checkPUses2(SliceC & S1, SliceC::iterator SLPUSE, SliceC::iterator
          SLPUSEREF);

          SliceC::iterator find_D1_Node(SliceC & S1, SliceC::iterator SLPUSE, Sli-
70      ceC::iterator SLNODE);

          void addAllD1Nodes(SliceC & S1, SliceC::iterator SLNODE, int posGatter, int
          D2_RefNr);

          SliceC::iterator findLastPUse(SliceC & S1, int SliceNr);

75          SliceC::iterator findLastPUse2(SliceC & S1, int SliceNr, int RefNr);

```

```

        void checkLoopOrCond(SliceC & S1, SliceC::iterator SL1, SliceC::iterator
SLORIG, int Pos1);

        void buildIn_D2_Part(SliceC & S1, int posOR_D2, SliceC::iterator SL1, Sli-
5 ceC::iterator SLPUSE, int KnotenNrD2);

        int findD2Node(SliceC & S1, SliceC::iterator PUSE);

        int buildInIfTree(SliceC & S1, SliceC::iterator SLIF, SliceC::iterator
10 SLORIG, int pos1);

        SliceC::iterator lookForNextPUse(SliceC & S1, SliceC::iterator SNODE, Sli-
ceC::iterator SLPUSE);

15 void build_IFKF_Part(SliceC & S1, int posKFOR, SliceC::iterator SLPUSE);

        void build_A1_Part(SliceC & S1, int posIFDFOR, SliceC::iterator SLIF, Sli-
ceC::iterator SLORIG);

20 void buildD2_IFKF_Part(SliceC & S1, int posKFOR, SliceC::iterator SLPUSE);

        void buildD2_A1_Part(SliceC & S1, int posIFDFOR, SliceC::iterator SLIF, Sli-
ceC::iterator SLPUSE);

25 int buildD2InIfTree(SliceC & S1, SliceC::iterator SLIF, SliceC::iterator
SLORIG, int pos1);

        void buildLoopKF_Part(SliceC & S1, int posAND_KF, int posLOOP_KF, int posD2,
SliceC::iterator SLOOP);

30 int buildIn_LoopKF_ANDPart(SliceC & S1, int posAND_KF, SliceC::iterator
SLPUSE);

        void buildIn_LoopKF_ORPart(SliceC & S1, int posAND_KF, SliceC::iterator SL1,
35 int KnotenNrD2);

};

40 #endif

#ifdef _uwgkanteC
#define _uwgkanteC

45 #include <iostream>

using namespace std;

class uwgkanteC {

50 public:

        short int eingefalter;
        int Number;

55        uwgkanteC() {}

        uwgkanteC(int _Number) { Number = _Number; };

        bool operator == (const uwgkanteC& other) const
60 == other.Number); }; { return (Number

        bool operator != (const uwgkanteC& other) const
65 == other); }; { return !(*this

        bool operator < (const uwgkanteC& other) const
70 == other.Number); }; { return (Number

        bool operator > (const uwgkanteC& other) const
75 == other.Number); }; { return (Number

        friend ostream& operator << ( ostream& os, const uwgkanteC& Node);

#endif

```



```

5  #ifndef _uwgknotenC
   #define _uwgknotenC
   #include <iostream>
10  const int maxKnotentextLength=128;
   const int maxBemerkLength=1000;
   const int maxWahrVarLength=16;
15  typedef struct ww {
       short int Wert;
       struct ww *next;
       } TMCSWert;
20  typedef struct el {
       struct ftgatter* unabElement;
       struct ww *next;
       struct el *nextel;
25  } TMinSet;

   typedef enum {AND, OR, CAUSE, EFFECT, NOT} NodeIdentT;

   using namespace std;
30  class uwgknotenC {
public:
35         static int DummyNr;
           int GatterIdent;
           int outof;
           int CauseNr;
           char gattertext[maxKnotentextLength];
40         char gatterbemerkung[maxBemerkLength];
           double wahrsch;
           char wahrvar[maxWahrVarLength];
           NodeIdentT NodeIdent;

45         uwgknotenC() { NodeIdent = EFFECT;
                           CauseNr = 0;
                           strcpy (gatter-
text, "-- unknown --");
                           strcpy (gatter-
50         bemerkung, "--keine--");
                           GatterIdent =
                           DummyNr++; };

           uwgknotenC(NodeIdentT _NodeIdent)
55         _NodeIdent; { NodeIdent =
                           CauseNr = 0;

           strcpy(gattertext, "-- unknown --");
60         strcpy(gatterbemerkung, "--keine--");
                           GatterIdent =
                           DummyNr++; };

           uwgknotenC(NodeIdentT _NodeIdent, int _CauseNr/*, char _gattertext[]*/)
65         _NodeIdent; { NodeIdent =
                           CauseNr =
                           _CauseNr;

70         strcpy(gattertext, "-- unknown --");
           strcpy(gatterbemerkung, "--keine--");

           /*strcpy(gattertext, _gattertext);*/
75         DummyNr++; };
                           GatterIdent =

```

41

```

    uwgknotenC(NodeIdentT _NodeId, int _CauseNr, char _gattertext[], char
_gatterbemerkung[])
{ NodeIdent =
5  _NodeId;
  _CauseNr;
  strcpy(gattertext, "-- unknown --");
10  strcpy(gattertext, _gattertext);
  strcpy(gatterbemerkung, _gatterbemerkung);
  DummyNr++; };
15  GatterIdent =
    uwgknotenC(NodeIdentT _NodeId, int _CauseNr, char _gattertext[])
{ NodeIdent =
  _NodeId;
  CauseNr =
20  _CauseNr;
  /*strcpy (gat-
    tertext, "-- unknown --");*/
  strcpy(gattertext, _gattertext);
25  strcpy(gatterbemerkung, "--keine--");
  DummyNr++; };
  GatterIdent =
30  uwgknotenC(NodeIdentT _NodeId, char _gattertext[], char
_gatterbemerkung[])
{ NodeIdent.=
  _NodeId;
  CauseNr = 0;
35  strcpy(gattertext, _gattertext);
  strcpy(gatterbemerkung, _gatterbemerkung);
  DummyNr++; };
  GatterIdent =
40  uwgknotenC(NodeIdentT _NodeId, char _gattertext[])
{ NodeIdent =
  _NodeId;
  CauseNr = 0;
45  strcpy(gattertext, _gattertext);
  strcpy(gatterbemerkung, "--keine--");
50  DummyNr++; };
  GatterIdent =
55  int getCauseNr()
{ return CauseNr; };
  int getGatterIdent()
{ return GatterIdent; };
  NodeIdentT getNodeIdent() const
{ return NodeIdent; };
60  bool operator == (const uwgknotenC& other) const
{ return ((Cau-
seNr == other.CauseNr)
  & (NodeIdent ==
65  other.NodeIdent)
  &
  (strcmp(gattertext, other.gattertext) == 0)
  & (GatterI-
dent == other.GatterIdent)); };
70  bool operator != (const uwgknotenC& other) const
{ return !(*this
== other); };
  bool operator < (const uwgknotenC& other) const
{ return (Gatte-
75  rIdent < other.GatterIdent); };
  bool operator > (const uwgknotenC& other) const

```

```

rIdent > other.GatterIdent); };
{ return (Gatte-

5         friend ostream& operator << ( ostream& os, const uwgknotenC& Node);
};

#endif

10 #include "AS_GraphKante.h"
    □
    #include <iostream>

15         ostream& operator << ( ostream& os, const GraphKanteC & Kante){
            os << Kante.KantenTyp;
            return os;
        }

20 #include "AS_GraphNode.h"
    □
    #include <iostream>
    □

25 □
    □
        ostream& operator << ( ostream& os, const GraphNodeC& Node){
    □
        os << endl << Node.NodeNr;
        return os;
    }

30 }

35 #include "AS_Slice.h"
    □
    #include <iostream>
    □
    #include <iterator>
    □

40 #include <algorithm>
    □
    #include "uwggraph.h"
    □

45 □
    using namespace std;
    □

50 □
    □
    //F0: Funktion zur Auswahl der Variable, für die ein Fehlerbaum erstellt werden soll.
    □
    // Zurückgegeben wird ein Zeiger auf den ausgewählten Knoten.
55 KFGListC::iterator SliceC::defineVariableToSlice(KFGListC & L2) {
        int Number = 0;
        KFGListC::iterator KFGI = L2.begin();
        KFGListC::iterator VARDEFI = L2.begin();
        KFGUseListC::iterator USEIT;
60         while ( KFGI != L2.end()) {
            USEIT = KFGI->Uses.begin();
            if ( KFGI->getKFGNodeType() == OP ) {
                cout << KFGI->getKnotenNummer() << ": " << USEIT->getUse() <<
65         endl;;
            }
            KFGI++;
        }
        cout << "Geben Sie die Knotennummer ein und drücken Sie RETURN: ";
70         cin >> Number;
        while ( KFGI != L2.begin()) {
            if (KFGI->getKnotenNummer() == Number) {
                VARDEFI = KFGI;
            }
            KFGI--;
75         }
        if (VARDEFI->getKFGNodeType() == OP) {
            cout << "Ausgewählt wurde Knoten Nr.: " << VARDEFI->getKnotenNummer() <<
            endl;
        }
    }
}

```

```

        return VARDEFI;
    }
    else {
5      VARDEFI = L2.begin();
      cout << "Dieser Knoten ist kein Ausgabeknoten, das Programm wird beendet."
      << endl;
      exit(1);
      return VARDEFI;
10    }

//F1: Sucht P-Use Knoten, fügt ihn dem Slice hinzu und zieht eine KFK.
// input: Iterator auf den C-Use oder D-Use, KFG-Liste;
15 void SliceC::sliceForLoops(KFGListC & L2, KFGListC::iterator LOOPIT) {
    int HLevel;
    int check = 0;
    int KnotenNummer = 0;
    KFGListC::iterator HELPIT = LOOPIT;
20    while ((LOOPIT->getKFGNodeType() != LC && LOOPIT->getKFGNodeType() != IFC) ||
        (LOOPIT->getLevel() != HELPIT->getLevel())) {
        LOOPIT--;
    }
    KnotenNummer = LOOPIT->getNodeNr();
    check = checkForNodes(KnotenNummer);
    //insert(*LOOPIT,*HELPTIT,KFK);
    insert(*HELPTIT,*LOOPIT,KFK); //umgekehrte Richtung der Kante, damit jeder Knoten
    seine Vorgänger kennt.
30    //cout << "eingefügterKnotenFla: " << LOOPIT->getNodeNr() << endl;
    //cout << "eingefügterKnotenFla: " << HELPTIT->getNodeNr() << endl << endl;
    if (check == 0) {
        findAllDefs(L2, LOOPIT);
    }
    HLevel = LOOPIT->getLevel();
35    while (LOOPIT->getLevel() > 2) {
        check == 0;
        HELPTIT = LOOPIT;
        while ((LOOPIT->getKFGNodeType() != LC && LOOPIT->getKFGNodeType() != IFC)
40    || (LOOPIT->getLevel() >= HELPTIT->getLevel())) {
            LOOPIT--;
        }
        KnotenNummer = LOOPIT->getNodeNr();
        check = checkForNodes(KnotenNummer);
        //insert(*LOOPIT,*HELPTIT,KFK);
        insert(*HELPTIT,*LOOPIT,KFK); //umgekehrte Richtung der Kante, damit jeder
        Knoten seine Vorgänger kennt.
45    //cout << "eingefügterKnotenFlb: " << LOOPIT->getNodeNr() << endl;
        //cout << "eingefügterKnotenFlb: " << HELPTIT->getNodeNr() << endl << endl;
        if (check == 0) {
            findAllDefs(L2, LOOPIT);
        }
        HLevel = LOOPIT->getLevel();
    }
55 }

//F2: Sucht alle D-Uses zu einem P-Use, fügt diese dem Slice hinzu und zieht DFK.
// input: Iterator auf den Knoten mit der P-Use Liste, KFG-Liste;
60 void SliceC::findAllDefs(KFGListC & L1, KFGListC::iterator ITER) {
    int AktLevel;
    int dummyIf = 0;
    int KnotenNummerEnde = 0;
    AktLevel = ITER->getLevel();
    KFGListC::iterator HELPTIT = ITER;
65    KFGListC::iterator HELPIF = ITER;
    KFGListC::iterator HELPLOOF = ITER;
    KFGListC::iterator UPPERLIMIT = ITER;
    KFGP_UseListC::iterator PUSEIT;
    PUSEIT = ITER->P_Uses.begin();
70    KFGDefListC::iterator DEFIT1;
    if (ITER->getKFGNodeType() == LC) {
        while ((HELPLOOF->getKFGNodeType() != EL) || (HELPLOOF->getLevel() != Akt-
Level)) {
            HELPLOOF++;
75        }
        HELPTIT = HELPLOOF;
        KnotenNummerEnde = HELPLOOF->getNodeNr();
    }
}

```

44

```

while (PUSEIT != ITER->P_Uses.end()) {
    UPPERLIMIT = findUpperLimit(L1, ITER, PUSEIT);
    HELPLOOP = findLowerLimit(L1, ITER, UPPERLIMIT);
    //insert (*UPPERLIMIT,*ITER,DFK);
    insert(*ITER,*UPPERLIMIT,DFK); //umgekehrte Richtung der Kante, damit jeder
5   Knoten seine Vorgänger kennt.
    //cout << "eingefügterKnotenF2a: " << UPPERLIMIT->getNodeNr() << endl;
    //cout << "eingefügterKnotenF2a: " << ITER->getNodeNr() << endl;
    while (HELPLOOP != UPPERLIMIT) {
10        HELPLOOP--;
        if (HELPLOOP->getKFGNodeType() == EE1) {
            dummyIf = 1;
        }
        if ((HELPLOOP->getKFGNodeType() == ETh) && (HELPLOOP->getLevel() <
15   ITER->getLevel()) && (dummyIf == 0)) {
            HELPIF = HELPLOOP;
            while ((HELPLOOP->getKFGNodeType() != IFC) || (HELPLOOP-
>getLevel() != HELPIF->getLevel())) {
20                HELPLOOP--;
            }
            dummyIf = 0;
        }
        if (HELPLOOP->getKFGNodeType() == NO) {
25            DEFIT1 = HELPLOOP->Defs.begin();
            while (DEFIT1 != HELPLOOP->Defs.end()) {
                if (strcmp(PUSEIT->getUse(),DEFIT1->getDef()) == 0) {
                    //insert(*HELPLOOP,*ITER,DFK);
                    insert(*ITER,*HELPLOOP,DFK); //umgekehrte Rich-
30   tung der Kante, damit jeder Knoten seine Vorgänger kennt.
                    defInLoop(L1,HELPLOOP);
                    //cout << "eingefügterKnotenF2b: " << HELPLOOP-
>getNodeNr() << endl;
                    //cout << "eingefügterKnotenF2b: " << ITER-
35   >getNodeNr() << endl << endl;
                    if (!HELPLOOP->Uses.empty()) {
                        findDefToCUse(L1, HELPLOOP);
                    }
                }
                DEFIT1++;
            }
        }
        PUSEIT++;
45   }
    KFKPUseToCUse(ITER,KnotenNummerEnde);
}

//F5: Funktion sucht den ersten def oberhalb eines p-uses oder c-uses und
// liefert einen Iterator auf diese obere Grenze bei der Suche aller defs
// zurück.
KFGListC::iterator SliceC::findUpperLimit(KFGListC & L1, KFGListC::iterator ITER,
55   KFGF_UseListC::iterator PUSEIT) {
    int dummy = 0;
    int dummy1 = 0;
    int dummyIf = 0;
    int AktLevel = 0;
    KFGListC::iterator HELPDEF = ITER;
    KFGListC::iterator HELPLOOP = ITER;
    KFGListC::iterator HELPIF = ITER;
    KFGListC::iterator HELPIT = ITER;
    KFGListC::iterator HELPDEFRETURN = ITER;
    KFGDefListC::iterator DEFIT;
60   AktLevel = ITER->getLevel();
    //finden des ersten defs oberhalb des p-use.
    while ((HELPDEF != L1.begin()) && (dummy != 1)) {
        HELPDEF--;
        if (HELPLOOP->getKFGNodeType() == EE1) {
70            dummyIf = 1;
        }
        if ((HELPDEF->getKFGNodeType() == ETh) && (HELPDEF->getLevel() < ITER-
>getLevel()) && (dummyIf == 0)) {
            HELPIF = HELPDEF;
            while ((HELPDEF->getKFGNodeType() != IFC) || (HELPDEF->getLevel() !=
75   HELPIF->getLevel())) {
                HELPDEF--;
            }
        }
    }
}

```

```

dummyIf = 0;
}
if (HELPDEF->getKFGNodeType() == NO) {
    DEFIT = HELPDEF->Defs.begin();
    if (strcmp(DEFIT->getDef(), PUSEIT->getUse()) == 0) {
        HELPDEFRETURN = HELPDEF;
        dummy = 1;
    }
}
}
if ((HELPDEF == L1.begin()) && (HELPDEF->getKFGNodeType() == NO)) {
    DEFIT = HELPDEF->Defs.begin();
    if (strcmp(DEFIT->getDef(), PUSEIT->getUse()) == 0) {
        HELPDEFRETURN = HELPDEF;
        dummy = 1;
    }
}
//finden der äußersten Schleife, falls diese überhaupt existiert.
if (ITER->getLevel() > 2) {
    while ((HELPIT != L1.end()) && (dummy1 != 1)) {
        HELPIT++;
        if ((HELPIT->getKFGNodeType() == EL) && (HELPIT->getLevel() < Akt-
Level)) {
            HELPLOOP = HELPIT;
            if (HELPIT->getLevel() > 2) {
                AktLevel = HELPIT->getLevel();
            }
            else {
                dummy1 = 1;
            }
        }
    }
    HELPIT = HELPLOOP;
    //gehe zum Anfang der äußersten Schleife:
    while ((HELPIT->getKFGNodeType() != LC) || (HELPIT->getLevel() != HELPLOOP-
>getLevel())) {
        HELPIT--;
    }
    //wenn der gefundene def-Knoten außerhalb der äußersten Schleife liegt, und
    //sein ScopeLevel größer ist als das der äußersten Schleife, suche, ob es
    //noch einen weiteren def-Knoten weiter oben gibt.
    if (HELPDEF->getNodeNr() > HELPIT->getNodeNr()) {
        if (HELPDEF->getLevel() >= HELPIT->getLevel()) {
            dummy = 0;
            while ((HELPDEF != L1.begin()) && (dummy != 1)) {
                HELPDEF--;
                DEFIT = HELPDEF->Defs.begin();
                if (strcmp(DEFIT->getDef(), PUSEIT->getUse()) == 0) {
                    HELPDEFRETURN = HELPDEF;
                    dummy = 1;
                }
            }
        }
    }
}
return HELPDEFRETURN;
}

//F6: Funktion sucht, abhängig vom oberen Ende, das untere Ende des Bereichs, indem
// die defs gesucht werden müssen.
// Zurückgeliefert wird ein Iterator auf die untere Grenze.
KFGListC::iterator SliceC::findLowerLimit(KFGListC & L1, KFGListC::iterator ITER,
KFGListC::iterator UPPERLIMIT) {
    int dummy1 = 0;
    int AktLevel = 0;
    KFGListC::iterator HELPIT = ITER;
    KFGListC::iterator HELPLOOP = ITER;
    KFGListC::iterator LOWERLIMIT = ITER;
    AktLevel = ITER->getLevel();
    if (ITER->getKFGNodeType() == LC) {
        while ((HELPLoop->getKFGNodeType() != EL) || (HELPLoop->getLevel() != Akt-
Level)) {
            HELPLOOP++;
        }
    }
    //finden des Endes der äußersten Schleife.

```

```

    if (ITER->getLevel() > 2) {
        while ((HELPIT != L1.end()) && (dummy1 != 1)) {
            HELPIT++;
            if ((HELPIT->getKFGNodeType() == EL) && (HELPIT->getLevel() < Akt-
5   Level)) {
                HELPLOOP = HELPIT;
                if (HELPIT->getLevel() > 2) {
                    AktLevel = HELPIT->getLevel();
                }
                else {
                    dummy1 = 1;
                }
            }
        }
        HELPIT = HELPLOOP; //HELPLoop zeigt auf das Ende der ä-
        Bersten Schleife, ansonsten auf den p-use.
        while ((HELPIT->getKFGNodeType() != LC) || (HELPIT->getLevel() != HELPLOOP-
        >getLevel())) {
            HELPIT--;
20   } //HELPLoop zeigt auf den
        Anfang der äußersten Schleife.
        if (HELPIT->getNodeNr() < UPPERLIMIT->getNodeNr()) {
            LOWERLIMIT = HELPLOOP; //falls die obere Grenze außerhalb der Schlei-
25   fen liegt.
        }
        else {
            AktLevel = UPPERLIMIT->getLevel();
            while ((HELPLoop->getKFGNodeType() != EL) || (HELPLoop->getLevel() <=
30   AktLevel)) {
                HELPLOOP--;
            }
            LOWERLIMIT = HELPLOOP; //falls die obere Grenze innerhalb der Schleife
            liegt.
        }
        else {
            LOWERLIMIT = HELPLOOP;
        }
        return LOWERLIMIT;
40   }

//F3: Funktion zieht eine KFK vom F-Use zu allen C-Uses, die sich in derselben
// Schleifen- bzw. Verzweigungsebene befinden.
// input: Iterator auf P-Use Knoten.
void SliceC::KFKPUseToCUse(KFGListC::iterator PUSEIT, int SchleifenEnde) {
    SliceC::iterator SL1 = begin();
    SliceC::iterator SLHELP = begin();
50   KFGListC::iterator ITER = PUSEIT;
    KFGUseListC::iterator USEIT;
    KFGP_UseListC::iterator PUSEIT2;
    KFGUseListC::iterator CUSEIT2;
    int KnotenLevel;
    int KnotenNummer;
    KnotenNummer = PUSEIT->getNodeNr();
    KnotenLevel = PUSEIT->getLevel();
    while ((*SL1).first.getNodeNr() != KnotenNummer)
        SL1++;
60   SLHELP = SL1;
    SL1 = begin();
    while (SL1 != end()) {
        if ((KnotenNummer > (*SL1).first.getNodeNr()) && (SchleifenEnde <
65   (*SL1).first.getNodeNr())) {
            if ((*SL1).first.getKFGNodeType() == NO) && ((*SL1).first.getLevel()
            == KnotenLevel)) {
                //insert((*SLHELP).first, (*SL1).first, KFK);
                insert((*SL1).first, (*SLHELP).first, KFK); //umgekehrte Rich-
70   tung der Kante, damit jeder Knoten seine Vorgänger kennt.
                //cout << "eingefügterKnotenF3: " <<
                (*SL1).first.getNodeNr() << endl;
                //cout << "eingefügterKnotenF3: " <<
                (*SLHELP).first.getNodeNr() << endl;
            }
        }
        SL1++;
    }
}

```

```

//F4: Funktion sucht, wenn der Knoten neben den Defs noch c-uses enthält, zu diesen
// c-uses alle defs und zieht DFK vom gefundenem def zum c-use.
5 void SliceC::findDefToCUse(KFGListC & L1, KFGListC::iterator ITER) {
    int HLevel;
    int check = 0;
    int dummy = 0;
    int dummyIf = 0;
10    int KnotenNummer = 0;
    KFGListC::iterator HELPIT = ITER;
    KFGListC::iterator HELPLOOP = ITER;
    KFGListC::iterator HELPIF = ITER;
    KFGListC::iterator LOWERLIMIT = ITER;
15    KFGListC::iterator UPPERLIMIT = ITER;
    KFGDefListC::iterator DEFIT1;
    KFGUseListC::iterator USEIT1;
    USEIT1 = ITER->Uses.begin();
    HLevel = ITER->getLevel();
20    while (USEIT1 != ITER->Uses.end()) {
        UPPERLIMIT = findUpperLimitFromCUse(L1, ITER, USEIT1);
        HELPIT = findLowerLimitFromCUse(L1, ITER, UPPERLIMIT);
        UPPERLIMIT--;
        //cout << "eingefügterKnotenF4a: " << UPPERLIMIT->getNodeNr() << endl;
        //cout << "eingefügterKnotenF4a: " << ITER->getNodeNr() << endl;
        while (HELPIF != UPPERLIMIT) {
            if (HELPLLOOP->getKFGNodeType() == EE1) {
                dummyIf = 1;
            }
30            if ((HELPIF->getKFGNodeType() == ETh) && (HELPIF->getLevel() <= ITER-
>getLevel()) && (dummyIf == 0)) {
                HELPIF = HELPIT;
                while ((HELPIF->getKFGNodeType() != IFC) || (HELPIF-
>getLevel() != HELPIF->getLevel())) {
35                    HELPIT--;
                }
                dummyIf = 0;
            }
            if (HELPIF->getKFGNodeType() == NO) {
                DEFIT1 = HELPIT->Defs.begin();
                if (strcmp(DEFIT1->getDef(),USEIT1->getUse()) == 0) {
                    if (!HELPIF->Uses.empty()) {
                        KnotenNummer = HELPIT->getNodeNr();
                        check = checkForNodes(KnotenNummer);
45                    }
                    //insert(*HELPIF,*ITER,DFK);
                    insert(*ITER,*HELPIF,DFK); //umgekehrte Richtung der
Kante, damit jeder Knoten seine Vorgänger kennt.
                    defInLoop(L1,HELPIF);
50                    //cout << "eingefügterKnotenF4b: " << HELPIT-
>getNodeNr() << endl;
                    //cout << "eingefügterKnotenF4b: " << ITER-
>getNodeNr() << endl << endl;
                    if (check == 0) {
                        findDefToCUse(L1,HELPIF);
55                    }
                }
            }
            HELPIT--;
60        }
        USEIT1++;
    }
}

65

//F8: Funktion sucht die untere Grenze, von der an die defs zu einem c-use gesucht
// werden dürfen.
KFGListC::iterator SliceC::findLowerLimitFromCUse(KFGListC & L1, KFGListC::iterator ITER,
70 KFGListC::iterator UPPERLIMIT) {
    int HLevel;
    int dummy = 0;
    KFGListC::iterator HELPIT = ITER;
    KFGListC::iterator HELPLOOP = ITER;
75    KFGListC::iterator LOWERLIMIT = ITER;
    KFGDefListC::iterator DEFIT1;
    KFGUseListC::iterator USEIT1;
    USEIT1 = ITER->Uses.begin();

```



```

HLevel = ITER->getLevel();
//Ist der c-use Knoten in einer Schleife?
if (ITER->getLevel() >= 2) {
    while ((HELPIT != L1.end()) && (dummy != 1)) {
        HELPIT++;
        if ((HELPIT->getKFGNodeType() == EL) && (HELPIT->getLevel() <=
5      HLevel)) {
            HELPLOOP = HELPIT;
            if (HELPIT->getLevel() > 2) {
10              HLevel--;
            }
            else {
                dummy = 1;
            }
        }
    }
    //Der Iterator HELPLOOP zeigt jetzt auf das Ende der äußersten, den Knoten umgeben-
20    den
    //Schleife, falls es diese gibt, ansonsten zeigt der Iterator auf den c-use-Knoten
    selbst.
    if (HELPLOOP != ITER) {
        HELPIT = HELPLOOP;
        //Iterator zeigt auf das Ende der äußersten
25    Schleife.
        while ((HELPIT->getKFGNodeType() != LC) || (HELPIT->getLevel() != HELPLOOP-
        >getLevel())) {
            HELPIT--;
        }
        if (HELPIT->getNodeNr() < UPPERLIMIT->getNodeNr()) {
30          LOWERLIMIT = HELPLOOP; //obere Grenze außerhalb der äußersten Schlei-
          fe;
        }
        else if (ITER->getLevel() == UPPERLIMIT->getLevel()) {
35          LOWERLIMIT = ITER; //obere Grenze mit dem c-use in dersel-
          ben Schleife;
        }
        else if (ITER->getLevel() < UPPERLIMIT->getLevel()) {
40          LOWERLIMIT = HELPLOOP; //obere Grenze in derselben äußeren Schleife,
          aber
          //innerhalb in einer
          anderen, vor dem c-use liegenden Schleife;
        }
        else {
            HLevel = UPPERLIMIT->getLevel();
            while ((HELPLOOP->getKFGNodeType() != EL) || (HELPLOOP->getLevel() <=
45          HLevel)) {
                HELPLOOP--;
            }
            LOWERLIMIT = HELPLOOP; //obere Grenze in einer niedrigeren Schleife
50          als der c-use;
        }
    }
    else {
        LOWERLIMIT = ITER; //keine äußere Schleife vorhanden.
55    }
    return LOWERLIMIT;
}

//F7: Funktion sucht die obere Grenze, bis zu der defs zu einem c-use gesucht werden
60 // dürfen. Die Funktionsweise ist ähnlich entsprechenden Funktion für p-uses.
KFGListC::iterator SliceC::findUpperLimitFromCUse(KFGListC & L1, KFGListC::iterator ITER1,
KFGUseListC::iterator USEIT) {
    int dummy = 0;
    int dummy1 = 0;
65    int dummyIf = 0;
    int AktLevel = 0;
    KFGListC::iterator HELPDEF = ITER1;
    KFGListC::iterator HELPLOOP = ITER1;
    KFGListC::iterator HELPIT = ITER1;
    KFGListC::iterator HELPIF = ITER1;
    KFGListC::iterator HELPDEFRETURN = ITER1;
    KFGDefListC::iterator DEFIT;
    AktLevel = ITER1->getLevel();
70    //cout << "bearbeitender Knoten: " << ITER1->getNodeNr() << endl;
    //finden des ersten defs oberhalb des use.
    do {
        if (HELPLOOP->getKFGNodeType() == EE1) {
            dummyIf = 1;
75
        }
    }

```

```

    }
    if ((HELPDEF->getKFGNodeType() == ETh) && (HELPDEF->getLevel() <= ITER1-
>getLevel()) && (dummyIf == 0)) {
5      HELPIF = HELPDEF;
      while ((HELPDEF->getKFGNodeType() != IFC) || (HELPDEF->getLevel() !=
HELPIF->getLevel())) {
          HELPDEF--;
      }
      dummyIf = 0;
10    }
    if (HELPDEF->getKFGNodeType() == NO) {
        DEFIT = HELPDEF->Defs.begin();
        if (strcmp(DEFIT->getDef(), USEIT->getUse()) == 0) {
15          HELPDEFRETURN = HELPDEF;
          if (HELPDEF != ITER1) {
              dummy = 1;
          }
        }
    }
    }
20    HELPDEF--;
    } while ((HELPDEF != L1.begin()) && (dummy != 1));
    if ((HELPDEF == L1.begin()) && (HELPDEF->getKFGNodeType() == NO)) {
        DEFIT = HELPDEF->Defs.begin();
        if (strcmp(DEFIT->getDef(), USEIT->getUse()) == 0) {
25          HELPDEFRETURN = HELPDEF;
        }
    }
    HELPDEF = HELPDEFRETURN;
30    //cout << "erster def: " << HELPDEFRETURN->getNodeNr() << endl;
    //finden der äußersten Schleife, falls diese überhaupt existiert.
    if (ITER1->getLevel() >= 2) {
        while ((HELPIT != L1.end()) && (dummy1 != 1)) {
            HELPIT++;
            if ((HELPIT->getKFGNodeType() == EL) && (HELPIT->getLevel() <= Akt-
35 Level)) {
                HELPLOOP = HELPIT;
                if (HELPIT->getLevel() > 2) {
                    AktLevel--;
                }
                else {
                    dummy1 = 1;
                }
            }
        }
45    HELPIT = HELPLOOP;
    //cout << "Ende der äußersten Schleife: " << HELPIT->getNodeNr() << endl;
    //gehe zum Anfang der äußersten Schleife, wenn es diese gibt;
    if (HELPLOOP->getKFGNodeType() == EL) {
        while ((HELPIT->getKFGNodeType() != LC) || (HELPIT->getLevel() !=
50 HELPLOOP->getLevel())) {
            HELPIT--;
        }
        //cout << "Anfang der äußeren Schleife: " << HELPIT->getNodeNr() <<
endl;
55    //wenn der gefundene def-Knoten außerhalb der äußersten Schleife
    //sein ScopeLevel größer ist als das der äußersten Schleife, suche,
    //noch einen weiteren def-Knoten weiter oben gibt.
    if (HELPDEF->getNodeNr() > HELPIT->getNodeNr()) {
        if (HELPDEF->getLevel() >= HELPIT->getLevel()) {
            dummy = 0;
            while ((HELPDEF != L1.begin()) && (dummy != 1)) {
65              HELPDEF--;
              DEFIT = HELPDEF->Defs.begin();
              if (strcmp(DEFIT->getDef(), USEIT->getUse()) ==
0) {
                  HELPDEFRETURN = HELPDEF;
                  dummy = 1;
70              }
            }
        }
    }
    //c-use innerhalb der äußersten Schleife;
    else {
75      if (HELPDEF->getLevel() > ITER1->getLevel()) {
          dummy = 0;
          while (HELPDEF != HELPIT) {

```

```

50
HELPDEF--;
DEFIT = HELPDEF->Defs.begin();
if (strcmp(DEFIT->getDef(),USEIT->getUse()) ==
5 0) {
HELPDEFRETURN = HELPDEF;
}
}
}
10 //c-use in keiner Schleife, aber in einer Verzweigung;
else {
if (HELPDEF->getLevel() > ITER1->getLevel()) {
15 dummy = 0;
while ((HELPDEF != L1.begin()) && (dummy != 1)) {
HELPDEF--;
DEFIT = HELPDEF->Defs.begin();
if (strcmp(DEFIT->getDef(),USEIT->getUse()) == 0) {
20 HELPDEFRETURN = HELPDEF;
dummy = 1;
}
}
}
}
25 //c-use in keiner Schleife;
else {
if (HELPDEF->getLevel() > ITER1->getLevel()) {
30 dummy = 0;
while ((HELPDEF != L1.begin()) && (dummy != 1)) {
HELPDEF--;
DEFIT = HELPDEF->Defs.begin();
if (strcmp(DEFIT->getDef(),USEIT->getUse()) == 0) {
35 HELPDEFRETURN = HELPDEF;
dummy = 1;
}
}
}
}
40 //cout << "UpperLimitFunktion: " << HELPDEFRETURN->getNodeNr() << endl;
return HELPDEFRETURN;
}

45 //F9: Funktion überprüft, ob ein Knoten bereits im Slice vorhanden ist. Wenn ja,
// wird eine eins zurückgeliefert, andernfalls eine null.
// input: Knotennummer des gesuchten Knotens;
// output: Integerwert 0 oder 1;
50 int SliceC::checkForNodes(int NodeNumber) {
int dummy = 0;
SliceC::iterator SL1 = begin();
while ((SL1 != end()) && (dummy != 1)) {
if ((*SL1).first.getNodeNr() == NodeNumber) {
55 dummy = 1;
}
SL1++;
}
if (dummy == 1) {
60 return 1;
}
else {
return 0;
}
}
65

//F10: Funktion untersucht, ob ein def in einer Schleife ist; wenn ja, wird die
// Funktion F1, die den p-use sucht, aufgerufen. Dort wird als erstes nach-
// gesehen, ob der gefundene p-use bereits im Slice vorhanden ist.
70 void SliceC::defInLoop(KFGListC & L1, KFGListC::iterator ITER) {
int dummyLoop = 0;
KFGListC::iterator HELPIT = ITER;
while ((HELPIT != L1.begin()) && (dummyLoop != 1)) {
75 HELPIT--;
if (((HELPIT->getKFGNodeType() == LC) || (HELPIT->getKFGNodeType() == IFC))
&& (HELPIT->getLevel() == ITER->getLevel())) {
sliceForLoops(L1,ITER);
}
}
}

```

```

dummyLoop = 1;
    }
}
5
//F11: Funktion bekommt den ausgewählten Startknoten für den Slice übermmittelt,
// untersucht, ob dieser Knoten in einer Schleife ist; wenn ja, wird zuerst
10 // die Funktion "sliceForLoops" aufgerufen, ansonsten sofort "findDefToCUse".
void SliceC::startBuildSlice(KFGListC & L1/*, KFGListC::iterator OUTPUTIT*/) {
    int dummyNode = 0;
    KFGListC::iterator STARTVARI = defineVariableToSlice(L1);
    KFGListC::iterator HELPOUT = STARTVARI;
15 while ((HELPOUT != L1.begin()) && (dummyNode != 1)) {
    HELPOUT--;
    if (((HELPOUT->getKFGNodeType() == LC) || (HELPOUT->getKFGNodeType() ==
IFC)) && (HELPOUT->getLevel() == STARTVARI->getLevel())) {
20 sliceForLoops(L1, STARTVARI);
    dummyNode = 1;
    }
}
//cout << "F11" << OUTPUTIT->getNodeNr() << endl;
25 findDefToCUse(L1, STARTVARI);
}

void SliceC::sliceAusgeben() {
30 ofstream Ziel("Slice2.slc");
    ostream_iterator<KFGListNodeC> POSIT(Ziel, "\n");
    ostream_iterator<KFGListNodeC> POSIT2(Ziel, "\n");
    SliceC::iterator SLC = begin();
35 while (SLC != end()) {
    *POSIT++ = (*SLC).first;
    SliceC::Nachfolger::iterator IT = (*SLC).second.begin();
    SliceC::Nachfolger::iterator ITEND = (*SLC).second.end();
    while (IT != ITEND) {
40 //a = (*IT).first;
        *POSIT++ = SLC[(*IT).first].first;
        *IT++;
    }
    ++SLC;
45 }
}

#include "KFGDef.h"
□
#include <iostream>
55 □
□
ostream& operator << ( ostream& os, const KFGDefC& Node){
□
    os << "DEF:          " << Node.Def << "          " << Node.ScopeLevelD;
60 □
    return os;
□
}

#include "KFGLineNode.h"
□
#include <iostream>
70 ostream& operator << ( ostream& os, const KFGLineNodeC& Node){
    os << Node.Name << " " << Node.LineNumber << endl;
    return os;
}
75 //Funktion, die aus KFGTokenList einen KFG baut.
□

```

```

#include <iostream>
□
#include "KFGList.h"
□
5  #include "KFGTokenList.h"
   #include "KFGUse.h"
   #include "KFGDef.h"
   #include "KFGProgList.h"
10  #include <algorithm>

void KFGListC::TokenList2KFGList(KFGTokenListC & L1) {
    int dummy;
    KFGTokenListC::iterator TLI = L1.end();
15  KFGP_UseListC::iterator P_USEI;
    KFGUseListC::iterator USEI;
    dummy = 0;
    //while (strcmp(TLI->getName(),"main") != 0) {
    while ((TLI != L1.begin()) && (dummy != 1)) {
20      if (TLI->getTokenNodeType() == N) {
          if (strcmp(TLI->getName(),"main") == 0) {
              dummy = 1;
          }
      }
      //TLI++;
      switch (TLI->getTokenNodeType()) {
      case PL:
          {
30              TLI--;
              KFGListNodeC hN1("NORMAL",NO,TLI->getScopeLevel(),TLI-
>getLineNumber());
              hN1.Defs.push_back(KFGDefC(TLI->getName(),TLI-
>getScopeLevel()));
              push_front(hN1);
35              TLI--;
          }
          break;
/*case IF:
    push_front(KFGListNodeC("IF",IFC,TLI->getScopeLevel()));
    TLI--;
    break;*/
    case BT:
        //push_front(KFGListNodeC("BT",BTh,TLI->getScopeLevel()));
        {
45            TLI--;
            //KFGP_UseListC::iterator P_USEI;
            KFGListNodeC hN1("IFCOND",IFC,TLI->getScopeLevel(),TLI-
>getLineNumber());
            while (TLI->getTokenNodeType() != IF) {
50                KFGUseC hUset(TLI->getName(),TLI->getScopeLevel());
                P_USEI =
                find(hN1.P_Uses.begin(),hN1.P_Uses.end(),hUset);
                if (P_USEI == hN1.P_Uses.end()) {
55                    hN1.P_Uses.push_back(hUset);
                    Anzahl_P_Uses++;
                }
                //hN1.P_Uses.push_back(KFGUseC(TLI->getName(),TLI-
>getScopeLevel()));
                TLI--;
60            }
            push_front(hN1);
            Entscheidungen++;
        }
        break;
65    case ET:
        push_front(KFGListNodeC("ENDTHEN",ETH,TLI->getScopeLevel()));
        TLI--;
        break;
    case BE:
70        push_front(KFGListNodeC("BEGINELSE",BE1,TLI->getScopeLevel()));
        TLI--;
        break;
    case EE:
75        push_front(KFGListNodeC("ENDELSE",EE1,TLI->getScopeLevel()));
        TLI--;
        break;
/*case WHILE:
    push_front(KFGListNodeC("WHILE",LC,TLI->getScopeLevel()));

```

```

        TLI--;
        break;*/
    case BW:
        //push_front(KFGListNodeC("BW",BL,TLI->getScopeLevel()));
5         {
            TLI--;
            //KFGP_UseListC::iterator P_USEI;
            KFGListNodeC hN1("LOOPCOND",LC,TLI->getScopeLevel(),TLI-
10         >getLineNumber());
            while (TLI->getTokenNodeType() != WHILE) {
                KFGUseC hUset(TLI->getName(),TLI->getScopeLevel());
                P_USEI =
find(hN1.P_Uses.begin(),hN1.P_Uses.end(),hUset);
15         if (P_USEI == hN1.P_Uses.end()) {
                hN1.P_Uses.push_back(hUset);
                Anzahl_P_Uses++;
            }
            //hN1.P_Uses.push_back(KFGUseC(TLI->getName(),TLI-
20         >getScopeLevel()));
            TLI--;
        }
        push_front(hN1);
        Schleifenentscheidungen++;
    }
    break;
    case EW:
        push_front(KFGListNodeC("ENDLOOP",EL,TLI->getScopeLevel()));
        TLI--;
        break;
30     case BDOW:
        {
            TLI--;
            KFGListNodeC hN1("DOWHILELOOPCOND",DOWLC,TLI-
35         >getScopeLevel(),TLI->getLineNumber());
            while (TLI->getTokenNodeType() != DOWS) {
                KFGUseC hUset(TLI->getName(),TLI->getScopeLevel());
                P_USEI =
find(hN1.P_Uses.begin(),hN1.P_Uses.end(),hUset);
40         if (P_USEI == hN1.P_Uses.end()) {
                hN1.P_Uses.push_back(hUset);
                Anzahl_P_Uses++;
            }
            hN1.P_Uses.push_back(KFGUseC(TLI->getName(),TLI-
45         >getScopeLevel()));
            TLI--;
        }
        push_front(hN1);
    }
    break;
50     case DO:
        push_front(KFGListNodeC("DOWHILELOOP",DOWL,TLI->getScopeLevel()));
        TLI--;
        Schleifenentscheidungen++;
        break;
55     case EF:
        {
            //TLI--;
            int HLevel;
            HLevel = TLI->getScopeLevel();
            push_front(KFGListNodeC("ENDLOOP",EL,TLI->getScopeLevel()));
            //while (TLI->getTokenNodeType() != BF) {
            while ((TLI->getTokenNodeType() != BF) || (TLI-
60         >getScopeLevel() != HLevel)) {
                TLI--;
            }
            TLI--;
            if (TLI->getTokenNodeType() == PF)
                TLI--;
            KFGListNodeC hN1("NORMAL",NO,TLI->getScopeLevel(),TLI-
70         >getLineNumber());
            hN1.Defs.push_back(KFGDefC(TLI->getName(),TLI-
            >getScopeLevel()));
            Anzahl_Defs++;
            hN1.Uses.push_back(KFGUseC(TLI->getName(),TLI-
75         >getScopeLevel()));
            Anzahl_Uses++;
            push_front(hN1);
            //while (TLI->getTokenNodeType() != EF) {

```

54

```

>getScopeLevel() != HLevel)) {
    while ((TLI->getTokenNodeType() != EF) || (TLI-
        TLI++;
    }
    TLI--;
    Zaehlschleifenentscheidungen++;
    Schleifenentscheidungen++;
}
break;
case FOR_D:
{
    TLI--;
    KFGListNodeC hN1("LOOPCOND",LC,TLI->getScopeLevel(),TLI-
>getLineNumber());
    while (TLI->getTokenNodeType() != DEF) {
        KFGUseC hUset(TLI->getName(),TLI->getScopeLevel());
        P_USEI =
find(hN1.P_Uses.begin(),hN1.P_Uses.end(),hUset);
        if (P_USEI == hN1.P_Uses.end()) {
            hN1.P_Uses.push_back(hUset);
            Anzahl_P_Uses++;
        }
        //hN1.P_Uses.push_back(KFGUseC(TLI->getName(),TLI-
>getScopeLevel()));
        TLI--;
    }
    push_front(hN1);

    TLI--;
    KFGListNodeC hN2("NORMAL",NO,(TLI->getScopeLevel()-1),TLI-
>getLineNumber());
    hN2.Defs.push_back(KFGDefC(TLI->getName(),(TLI-
>getScopeLevel()-1)));
    Anzahl_Defs++;
    push_front(hN2);
    TLI--;
}
break;
case RET:
    //Nur eine vorläufige Implementierung; muß noch erweitert werden
    //für den Fall, daß nach dem RETURN eine Ausgabe folgt.
    push_front(KFGListNodeC("RETURN",RETURN,TLI->getScopeLevel()));
    TLI--;
    break;
case NL:
    TLI--;
    //Knoten erzeugen, der Defs und Uses enthält;
    if (TLI->getTokenNodeType() == N) {
        KFGListNodeC helpNode1("NORMAL",NO,TLI->getScopeLevel(),TLI-
50 >getLineNumber());
        TLI--;
        if (TLI->getTokenNodeType() == NL) {
            TLI++;
            helpNode1.Defs.push_back(KFGDefC(TLI->getName(),TLI-
55 >getScopeLevel()));
            Anzahl_Defs++;
            helpNode1.Uses.push_back(KFGUseC(TLI->getName(),TLI-
>getScopeLevel()));
            Anzahl_Uses++;
            TLI--;
        }
        else {
            TLI++;
            while (TLI->getTokenNodeType() != DEF) {
                KFGUseC hUset(TLI->getName(),TLI-
65 >getScopeLevel());
                USEI =
find(helpNode1.Uses.begin(),helpNode1.Uses.end(),hUset);
                if (USEI == helpNode1.Uses.end()) {
                    helpNode1.Uses.push_back(hUset);
                    Anzahl_Uses++;
                }
                //helpNode1.Uses.push_back(KFGUseC(TLI-
70 >getName(),TLI->getScopeLevel()));
                TLI--;
            }
            TLI--;

```

```

55
helpNode1.Defs.push_back(KFGDefC(TLI->getName(), TLI-
>getScopeLevel()));
5
    TLI--;
    Anzahl_Defs++;
    if (TLI->getTokenNodeType() == N) {
        helpNode1.Defs.pop_front();
        Anzahl_Defs--;
        //in
diesem Fall handelt es sich um ein Array und
10
        //daher muß der letzte Knoten wieder gelöscht werden.
        TLI++;
        //Dafür muß der gelöschte Knoten in die Use-Liste eingetragen werden.
        KFGUseC hUse2(TLI->getName(), TLI-
15
        >getScopeLevel());
        USEI =
        find(helpNode1.Uses.begin(), helpNode1.Uses.end(), hUse2);
        if (USEI == helpNode1.Uses.end()) {
            helpNode1.Uses.push_back(hUse2);
            Anzahl_Uses++;
20
        }
        TLI--;
        helpNode1.Defs.push_back(KFGDefC(TLI-
        helpNode1.Uses.push_back(KFGUseC(TLI-
25
        >getName(), TLI->getScopeLevel()));
        Anzahl_Defs++;
        Anzahl_Uses++;
        TLI--;
30
    }
    push_front(helpNode1);
}
//Knoten erzeugen, der nur Defs enthält;
35
else if (TLI->getTokenNodeType() == DEF) {
    TLI--;
    KFGListNodeC helpNode1("NORMAL", NO, TLI->getScopeLevel(), TLI-
    >getLineNumber());
    helpNode1.Defs.push_back(KFGDefC(TLI->getName(), TLI-
40
    >getScopeLevel()));
    push_front(helpNode1);
    Anzahl_Defs++;
    TLI--;
}
45
else if (TLI->getTokenNodeType() == PF) {
    TLI--;
    KFGListNodeC hN1("NORMAL", NO, TLI->getScopeLevel(), TLI-
    >getLineNumber());
    hN1.Defs.push_back(KFGDefC(TLI->getName(), TLI-
50
    >getScopeLevel()));
    hN1.Uses.push_back(KFGUseC(TLI->getName(), TLI-
    >getScopeLevel()));
    push_front(hN1);
    Anzahl_Defs++;
    Anzahl_Uses++;
55
}
//Knoten erzeugen, der Output-Variablen enthält;
else {
    TLI--;
    KFGListNodeC helpNode1("OUTPUT", OP, TLI->getScopeLevel(), TLI-
60
    >getLineNumber());
    helpNode1.Uses.push_back(KFGUseC(TLI->getName(), TLI-
    >getScopeLevel()));
    push_front(helpNode1);
    Anzahl_Uses++;
65
    TLI--;
}
    Anweisungen++;
    break;
70
default:
    TLI--;
}
    KnotenNummern();
    KFGListC::iterator KFG = begin();
    KnotenIdentifizierer(KFG);
    addLineInToList();
    Anzahl_Deklarationen = zaehleDeklarationen(L1);
75
}

```



```

5 void KFGListC::KnotenNummern() {
    int KnotenNr = 1;
    KFGListC::iterator KFG1 = begin();
    while (KFG1 != end()) {
        if ((KFG1->getKFGNodeType() == EL) || (KFG1->getKFGNodeType() == ETH) ||
10 (KFG1->getKFGNodeType() == BE1) || (KFG1->getKFGNodeType() == EE1)) {
            KFG1++;
        }
        else {
            KFG1->setKFGKnotenNummer(KnotenNr);
            KnotenNr++;
            KFG1++;
15
        }
    }
}

20 void KFGListC::KnotenIdentifizierer(KFGListC::iterator KFG) {
    int LOOPLevel = 0;
    int THENLevel = 0;
    int ELSELevel = 0;
25 KFGListC::iterator KFG1 = KFG;
    KFGListC::iterator LOOPIT = KFG;
    KFGListC::iterator IFIT = KFG;
    KFGListC::iterator ELSEIT = KFG;
    while (KFG1 != end()) {
30         switch(KFG1->getKFGNodeType()) {
            case LC:
                {
                    LOOPLevel = KFG1->getLevel();
                    KFG1++;
                    LOOPIT = KFG1;
                    while ((KFG1->getKFGNodeType() != EL) || (KFG1->getLevel() !=
35 LOOPLevel)) {
                        KFG1->setKnotenIdent(LLOOP);
                        KFG1++;
                    }
                    KnotenIdentifizierer(LOOPIT);
                }
                break;
            case IFC:
45                 {
                    THENLevel = KFG1->getLevel();
                    KFG1++;
                    IFIT = KFG1;
                    while ((KFG1->getKFGNodeType() != ETH) || (KFG1->getLevel()
50 != THENLevel)) {
                        KFG1->setKnotenIdent(THEN);
                        KFG1++;
                    }
                    KnotenIdentifizierer(IFIT);
                }
                break;
            case BE1:
60                 {
                    ELSELevel = KFG1->getLevel();
                    KFG1++;
                    ELSEIT = KFG1;
                    while ((KFG1->getKFGNodeType() != EE1) || (KFG1->getLevel()
65 != ELSELevel)) {
                        KFG1->setKnotenIdent(ELSE);
                        KFG1++;
                    }
                    KnotenIdentifizierer(ELSEIT);
                }
                break;
70             default:
                //cout << "Autsch" << endl;
                KFG1++;
            }
            //KFG1++;
75
        }
    }
}

```

```

void KFGListC::addLineInToList() {
    int zahl;
    char line[256];
    KFGProgListC LP;
5    ifstream datei("CodeLine.dat");
    if (!datei) {
        cout << "ERROR: Cannot open file 'CodeLine.dat'." << endl;
    }
    else {
10        while (!datei.eof()) {
            datei.getline(line, 255, '\n');
            for (int i=0; i<strlen(line); i++) {
                if (line[i] == '"') {
15                    line[i] = '\\';
                }
                if (line[i] == '{') {
                    line[i] = ';';
                }
            }
20            zahl = atoi(line);
            KFGListNodeC hknoten(line, zahl);
            LP.push_back(hknoten);
            //cout << line << " " << zahl << endl;
        }
25        addLineToKFG(LP);
    }
}

30 void KFGListC::addLineToKFG(KFGProgListC & LP) {
    int dummy = 0;
    char help[256];
    KFGListC::iterator KFG1 = end();
    KFGProgListC::iterator PROG1 = LP.end();
35    //KFG1--;
    while (KFG1 != begin()) {
        KFG1--;
        if ((KFG1->getKFGNodeType() == EL) || (KFG1->getKFGNodeType() == ETh) ||
40        (KFG1->getKFGNodeType() == BE1) || (KFG1->getKFGNodeType() == EE1)) {
            KFG1--;
        }
        while ((PROG1 != LP.begin()) && (dummy != 1)) {
            if (KFG1->getLineNr() == PROG1->getLineNumber()) {
45                strcpy(help, PROG1->getName());
                KFG1->setCodeLine(help);
                PROG1--;
                dummy = 1;
            }
            else{
50                PROG1--;
            }
        }
        PROG1 = LP.end();
        dummy = 0;
55    }
}

60 int KFGListC::zaehleDeklarationen(KFGTokenListC & TL) {
    int zaehler = 0;
    KFGTokenListC::iterator TListI = TL.end();
    while (strcmp(TListI->getName(), "main") != 0) {
        if (TListI->getTokenNodeType() == TD) {
65            zaehler++;
        }
        TListI--;
    }
    return zaehler;
70 }

//Funktion zur Ausgabe der KFGList auf den Bildschirm und in eine Datei "CFGList".
void KFGListC::ListeAusgeben() {
75    ofstream Ziell("CFGList.cfg");
    ostream_iterator<KFGListNodeC> Pos(Ziell, "\n");
    ostream_iterator<KFGDefC> PosD(Ziell, "\n");
    ostream_iterator<KFGUseC> PosU(Ziell, "\n");
    ostream_iterator<KFGUseC> PosF(Ziell, "\n");
}

```

```

KFGListC::iterator KLI = begin();
KFGDefListC::iterator DEF;
KFGUseListC::iterator USE;
KFGP UseListC::iterator P_USE;
5 Ziell << "START" << endl << endl;
while (KLI != end()) {
    Ziell << "Line " << KLI->getLineNr() << endl;
    Ziell << "Zeile " << KLI->getCodeLine() << endl;
    Ziell << "Knoten " << KLI->getKnotenNummer() << endl;
10 Ziell << "KnotenTyp " << KLI->getKnotenIdent() << endl;
    Ziell << KLI->getStatement() << " " << KLI->getLevel() << endl;
    /*Pos++ = *KLI;
    DEF = KLI->Defs.begin();
    USE = KLI->Uses.begin();
15 P_USE = KLI->P_Uses.begin();
    while (DEF != KLI->Defs.end()) {
        /*PosD++ = *DEF;
        Ziell << "DEF: " << DEF->getDef() << " " << DEF-
20 >getScopeLevelD() << endl;
        DEF++;
    }
    while (USE != KLI->Uses.end()) {
        /*PosU++ = *USE;
        Ziell << "USE: " << USE->getUse() << " " << USE-
25 >getScopeLevelU() << endl;
        USE++;
    }
    while (P_USE != KLI->P_Uses.end()) {
        /*PosP++ = *P_USE;
        Ziell << "P_USE: " << P_USE->getUse() << " " <<
30 P_USE->getScopeLevelU() << endl;
        P_USE++;
    }
    Ziell << endl;
35 KLI++;
}
Ziell << "STOP" << endl << endl;
Ziell << "BASISGROESSEN:" << endl;
Ziell << "Anzahl leerer Zweige: " << "0" << endl;
40 Ziell << "Anzahl Entscheidungen: " << Entscheidungen << endl;
Ziell << "Anzahl Anweisungen: " << Anweisungen << endl;
Ziell << "Anzahl atomarer Prädikate: " << Entscheidungen << endl;
Ziell << "Anzahl Prädikate: " << Entscheidungen << endl;
Ziell << "Anzahl atomarer Prädikate, die arithm. Relationen sind: " << "0" << endl;
45 Ziell << "Anzahl Schleifenentscheidungen: " << Schleifenentscheidungen << endl;
Ziell << "Anzahl nicht eingeschachtelter Schleifenentscheidungen: " << "0" << endl;
Ziell << "Anzahl Zählschleifenentscheidungen: " << Zaehlschleifenentscheidungen <<
endl;
50 Ziell << "Anzahl Deklarationen von strukturierten Datentypen: " << "0" << endl;
Ziell << "Anzahl Deklarationen: " << Anzahl_Deklarationen << endl;
Ziell << "Anzahl Realelement-Deklarationen: " << "0" << endl;
Ziell << "Anzahl Basistyp-Deklarationen: " << Anzahl_Deklarationen << endl;
Ziell << "Anzahl Defs: " << Anzahl_Defs << endl;
55 Ziell << "Anzahl Uses: " << Anzahl_Uses << endl;
Ziell << "Anzahl P-Uses: " << Anzahl_P_Uses << endl << endl;
basisgroessenInDatei();
}

60 void KFGListC::basisgroessenInDatei() {
    ofstream datei("Basisgroessen.bgd", ios::out);
    datei << "0" << endl;
    datei << Entscheidungen << endl;
    datei << Anweisungen << endl;
65 datei << Entscheidungen << endl;
    datei << Entscheidungen << endl;
    datei << "0" << endl;
    datei << Schleifenentscheidungen << endl;
    datei << "0" << endl;
70 datei << Zaehlschleifenentscheidungen << endl;
    datei << "0" << endl;
    datei << Anzahl_Deklarationen << endl;
    datei << "0" << endl;
    datei << Anzahl_Deklarationen << endl;
75 datei << "0" << endl;
    datei << Anzahl_Defs << endl;
    datei << Anzahl_Uses << endl;
    datei << Anzahl_P_Uses << endl;

```

```

    datei << "0" << endl;
    datei << "0" << endl;
    datei << "0" << endl;
    datei << "0" << endl;
5    datei << "0" << endl;
    datei << "0" << endl;
    datei << "0" << endl;
    datei << "0" << endl;
10    }

15

20

    #include "KFGListe.h"
25    □
    □
    KFGListeT KFGListe;
    □

30    □
    void KFGListeAusgeben(KFGListeT & L) {
    □
        KFGListeT::iterator I = L.begin();
35    □
        while(I != L.end())
            cout << *I++ << ' ';
            cout << " size() = " << L.size() << endl;
    }

40    #include "KFGListNode.h"
    □
    #include <iostream>
45    □
    □
    ostream& operator << ( ostream& os, const KFGListNodeC& Node){
    □
        os << endl << "KnotenNr:" << Node.KnotenNr << " Typ:" << Node.KnotenIdent <<
50    endl << Node.Statement << " Level:" << Node.Level;
        return os;
    }

55    int KFGListNodeC::DummyNodeNr=1;

    □
60    #include "KFGNode.h"
    □
    #include <iostream>
    □

65    □
    ostream& operator << ( ostream& os, const KFGNodeC& Node){
    □
        os << Node.Name << " " << Node.ScopeLevel;
        return os;
70    }

    #include "KFGTokenList.h"
    □
75    □
    □

```

60

```

void KFGTokenListC::KFGListeAusgeben() {
    ofstream Ziel("CFGTokenList");
    ostream_iterator<KFGTokenNodeC> oPos(Ziel, "\n");
    KFGTokenListC::iterator I = begin();
    while(I != end()) {
        /*oPos++ = I->getName();
        *oPos++ = *I;
        //cout << I->getName() << " " << I->getScopeLevel() << " " << I-
        >getTokenNodeType() << endl;
        I++;
    }
    cout << " size() = " << size() << endl;
}

#include "KFGTokenNode.h"
#include <iostream>

ostream& operator << ( ostream& os, const KFGTokenNodeC& Node){
    os << Node.Name << " " << Node.ScopeLevel << " " << Node.TokenNodeType << "
    << Node.LineNumber << endl;
    return os;
}

#include "KFGUse.h"
#include <iostream>

ostream& operator << ( ostream& os, const KFGUseC& Node){
    os << "USE: " << Node.Use << " " << Node.ScopeLevelU;
    return os;
}

/*
 * Main program to test C++ grammar and preprocessor
 */

#include "JLStr.h"
#include "tokens.h"
#include "DLGLexer.h"
#include "BufferedCPreParser.h"
#include "CPreToCPPBuffer.h"
#include "JLTokenBuffer.h"
#include "CPPParserSym.h"
#include "KFGTokenList.h"
#include "KFGList.h"
#include "graph.h"
#include "AS_Slice.h"
#include "uwggraph.h"
#include "KFGProgList.h"
#include <iostream>

static void usage(char* progname);

int main(int argc, char *argv[])
{
    // For reporting memory usage
    char *p = new char[100000];
    long heap1 = (long)(void*)p;

```

```

delete [] p;

// Parse command-line options
JLStr includePath;
JLStr definitions;
FILE *inputFile = stdin;
bool doTraceParse = false;
bool doTraceInclude = false;
bool doDump = false;
char *programe = *argv;

argc--;
argv++;

ofstream datei("CodeLine.dat", ios::trunc);

while (argc > 0)
{
    if (argv[0][0] == '-')
    {
        switch (argv[0][1])
        {
            case 'I':
                if (argv[0][2] != 0)
                {
                    // argument is part of "-I"
                    includePath += ';';
                    includePath += &argv[0][2];
                } else if (argc > 1)
                {
                    argc--;
                    argv++;
                    includePath += ';';
                    includePath = *argv;
                } else {
                    usage(programe);
                }
                break;
            case 'd':
                if (strcmp(argv[0], "-dump") == 0)
                {
                    doDump = true;
                }
                break;
            case 'D':
                if (argv[0][2] != 0)
                {
                    // argument is part of "-D"
                    definitions += ';';
                    definitions += &argv[0][2];
                } else if (argc > 1)
                {
                    // argument follows "-D"
                    argc--;
                    argv++;
                    definitions += ';';
                    definitions = *argv;
                } else {
                    usage(programe);
                }
                break;
            case 't':
                if (strcmp(*argv, "-traceParse") == 0)
                {
                    doTraceParse = true;
                } else if (strcmp(*argv, "-traceInclude") == 0)
                {
                    doTraceInclude = true;
                } else {
                    usage(programe);
                }
                break;
            default:
                usage(programe);
                break;
        }
    } else {
        // must be the input filename

```

```

    if (inputFile != stdin)
    {
        // already have one
        usage(progname);
5
    }
    FILE *fp = fopen(*argv, "r");
    if (fp == NULL)
    {
10
        fprintf(stderr, "%s: cannot open %s for input\n", progname, *argv);
        exit(0);
    }
    inputFile = fp;
}
15
argc--;
argv++;
}

//printf("%s\n%s\n%s\n",includePath,definitions,inputFile);

20
// Create input stream, lexer
DLGFileInput input(inputFile);
DLGLexer scanner(&input);
FastToken tok;
scanner.setToken(&tok);

25
// Create preprocessor parser. Note that the preprocessor parser has
// a built-in token buffer in the form of an input stack.
BufferedCPreParser preprocessor(&scanner);
preprocessor.init();

30
// set include path and defines directly for debugging
if (includePath.length() == 0)
{
35
    includePath = "c:\\devstudio\\vc\\include;\\msdev\\devstudio\\vc\\mfc\\include";
}
if (definitions.length() == 0)
{
    definitions =
40
        "_cplusplus;"
        "_DATE_=\"date\";"
        "_FILE_=\"filename\";"
        "_LINE_=\"1\";"
        "_TIME_=\"time\";"
45
        "_TIMESTAMP_=\"timestamp\";"
        "_M_IX86=400;"
        "_MSC_VER=1100;"
        "_WIN32;"
        "_INTEGRAL_MAX_BITS=64";
50
}

// Set preprocessor options
preprocessor.SetIncludePath(includePath);
preprocessor.SetDefinitions(definitions);

55
// Set options in parser to make it act standard with MS Extensions
preprocessor.SetOption(CPreParserImp::OptMSExtensions, true);
preprocessor.SetOption(CPreParserImp::OptBool, true);
preprocessor.SetOption(CPreParserImp::OptWCharT, true);
preprocessor.SetOption(CPreParserImp::OptPragmas, true);
60
preprocessor.SetOption(CPreParserImp::OptExpandPragmas, true);
preprocessor.SetOption(CPreParserImp::OptTrackInclude, doTraceInclude);

// Buffer to store tokens output by preprocessor
65
CPreToCPPBuffer pipe2(&preprocessor);

// Connect preprocessor to second token buffer so that the
// preprocessor can unilaterally squirt new tokens into the buffer
preprocessor.SetBuffer(&pipe2);

70
// Token buffer for the usual backtracking, etc.
JLTokenBuffer pipe3(&pipe2, CPPParserSym::ConstLLK);

// Create the C++ parser
CPPParserSym parser(&pipe3);

75
// Set parser options to look like MSVC++
parser.SetOption(CPPParserSym::OptPragmaMSPacking, true);
parser.SetOption(CPPParserSym::OptMSTypedefHack, true);

```

```

    //preprocessor.doTrace(doTraceParse);
    parser.doTrace(doTraceParse);

5    // Process top-level rule
    parser.init();

    KFGTokenListC L1;
    parser.translation_unit(L1);

10    // Close the input file stream
    if (inputFile != stdin)
    {
        fclose(inputFile);

15    }

    // For reporting memory usage
    // For reporting memory usage
    p = new char[100000];
20    long heap2 = (long)(void*)p;
    delete [] p;
    cout << "Approx heap usage before dump: " << (heap2 - heap1) << endl;

    if (doDump)
25    {
        // Dump the scope hierarchy
        cout << "Dump of scope hierarchy" << endl;
        parser.DumpScopes();
        cout << endl;

30    }

    // For reporting memory usage
    // For reporting memory usage
    p = new char[100000];
35    long heap3 = (long)(void*)p;
    delete [] p;
    cout << "Approx heap usage after dump: " << (heap3 - heap1) << endl;
    L1.KFGListeAusgeben();
    //L1.KFGTokenListToKFGList();

40    KFGListC L2;
    //KFGListC::iterator Iter;
    L2.TokenList2KFGList(L1);
    L2.ListeAusgeben();

45    SliceC Slice;
    Slice.startBuildSlice(L2);
    //Slice.buildTestGraph(L2);
    //cout << Slice;

50    uwggraphC uwggraph;
    //uwggraph.buildIfTree(Slice);
    uwggraph.startBuildFT(Slice);
    cout << uwggraph;

55    return 0;
}

static void usage(char* progame)
60 {
    fprintf(
        stderr,
        "usage: %s [-I directories] [-D definitions] [-traceParse] [ -traceInclude ] [-o
65 ofile] [ifile]\n"
        "    -I directories: semicolon-separated list of include directories\n"
        "    -D definitions: semicolon-separated list of definitions, like:\n"
        "        symbol1;symbol2=value2\n"
        "    -traceParse: output debugging information\n"
        "    -traceInclude: output trace of include stack\n"
70 "    -dump: dump type and scope hierarchy\n"
        "    ifile: name of input file (otherwise it uses stdin)\n",
        progame
    );
    exit(1);

75 }

#include <iostream>

```



```

#include <iterator>
#include <algorithm>
#include <fstream>
#include "uwggraph.h"

5 using namespace std;

SliceC::iterator uwggraphC::findOutputNode(SliceC & S1) {
10     int dummy = 0;
    SliceC::iterator SL1 = S1.begin();
    SliceC::iterator HELP = S1.begin();
    while ((SL1 != S1.end()) && (dummy != 1)) {
        if ( (*SL1).first.getKFGNodeType() == OP ) {
15             HELP = SL1;
            dummy = 1;
        }
        SL1 ++;
    }
    return HELP;
20 }

void uwggraphC::startBuildFT(SliceC & S1) {
25     int pos1 = 0;
    int Posil = 0;
    int SliceNr = 0;
    int LineNr = 0;
    int inLoop = 0;
30     SliceAusgeben(S1);
    char NodeText[128];
    char Bemerkung[1000];
    SliceC::iterator SL1 = findOutputNode(S1);
    SliceNr = (*SL1).first.getKnotenNumber();
35     LineNr = (*SL1).first.getLineNr();
    sprintf(NodeText, "Op%d", SliceNr);
    strcpy(Bemerkung, SL1->first.getCodeLine());
    //sprintf(Bemerkung, "Line %d", LineNr);
    SliceC::Nachfolger::iterator BEGIN = (*SL1).second.begin();
    SliceC::Nachfolger::iterator END = (*SL1).second.end();
40     while (BEGIN != END) {
        if (BEGIN->second == KFK) {
            uwgknotenC hknoten1(EFFECT, SliceNr, NodeText, Bemerkung);
45             Posil = insert(hknoten1);
            SliceC::iterator SLPUSE = findLastPUse(S1, SliceNr);
            checkLoopOrCond(S1, SLPUSE, SL1, Posil);
            inLoop = 1;
            BEGIN++;
        }
        else {
50             BEGIN++;
        }
    }
    if (inLoop == 0) {
55         uwgknotenC hknoten1(EFFECT, SliceNr, NodeText, Bemerkung);
        uwgknotenC hknoten2(OR, "OR");
        insert(hknoten1, hknoten2, 0);
        pos1 = size()-1;
        uwgknotenC hknoten3(CAUSE, SliceNr, NodeText, Bemerkung);
60         insert(hknoten2, hknoten3, 0);
        addFirstNodesToFT(S1, SliceNr, pos1);
    }
    check();
65 }

void uwggraphC::checkLoopOrCond(SliceC & S1, SliceC::iterator SLPUSE, SliceC::iterator
SLORIG, int Posil) {
70     int PosRet = 0;
    switch (SLPUSE->first.getKFGNodeType()) {
        case IFC:
            PosRet = buildInIfTree(S1, SLPUSE, SLORIG, Posil);
            //cout << "Test1a: " << SLHELP->first.getKnotenNumber() << endl;
            break;
75         case LC:
            buildInLoopTree(S1, Posil, SLPUSE, SLORIG);
            //cout << "Test1b: " << SLHELP->first.getKnotenNumber() << endl;
    }
}

```

```

        break;
    default:
        cout << "Ups" << endl;
    }
5  }

void uwggraphC::addFirstNodesToFT(SliceC & S1, int KnotenNr, int Pos1) {
10     int helpNr = 0;
        int ret = 0;
        int pos = 0;
        int KnotenNr1 = 0;
        int LineNr1 = 0;
        char NodeText[128];
15     char Bemerkung[1000];
        SliceC::iterator NODE = defineIterator(S1, KnotenNr);
        SliceC::Nachfolger::iterator BEGIN = NODE->second.begin();
        SliceC::Nachfolger::iterator END = NODE->second.end();
        while (BEGIN != END) {
20             helpNr = BEGIN->first;
            KnotenNr1 = S1[helpNr].first.getKnotenNummer();
            SliceC::iterator NODE1 = defineIterator(S1, KnotenNr1);
            if (NODE1->first.getLevel() > 1) {
25                 SliceC::iterator PUSE = findLastPUse(S1, KnotenNr1);
                switch (PUSE->first.getKFGNodeType()) {
                    case IFC:
                        ret = buildInIfTree(S1, PUSE, NODE1, Pos1);
                        break;
30                 case LC:
                        buildInLoopTree(S1, Pos1, PUSE, NODE1);
                        break;
                    default:
                        cout << "Something is wrong!" << endl;
35                 }
            }
            else {
                LineNr1 = S1[helpNr].first.getLineNr();
                sprintf(NodeText, "Op%d", KnotenNr1);
                //sprintf(Bemerkung, "Line %d", LineNr1);
                strcpy(Bemerkung, S1[helpNr].first.getCodeLine());
                uwgknotenC hknoten1(CAUSE, KnotenNr1, NodeText, Bemerkung);
                pos = insert(hknoten1);
                verbindeEcken(Pos1, pos, 0);
                addFirstNodesToFT(S1, KnotenNr1, Pos1);
45             }
            BEGIN++;
        }
    }

50

SliceC::iterator uwggraphC::findLastPUse(SliceC & S1, int SliceNr) {
55     int helpNr1 = 0;
        int SliceNr1 = 0;
        SliceC::iterator SL1 = defineIterator(S1, SliceNr);
        SliceC::iterator SLHELP = SL1;
        SliceC::Nachfolger::iterator BEGIN = (*SL1).second.begin();
        SliceC::Nachfolger::iterator END = (*SL1).second.end();
        while (BEGIN != END) {
60             if (BEGIN->second == KFK) {
                helpNr1 = BEGIN->first;
                SliceNr1 = S1[helpNr1].first.getKnotenNummer();
                SLHELP = defineIterator(S1, SliceNr1);
                SLHELP = findLastPUse(S1, SliceNr1);
65             }
            BEGIN++;
        }
        return SLHELP;
70    }

SliceC::iterator uwggraphC::findLastPUse2(SliceC & S1, int SliceNr, int RefNr) {
75     int helpNr1 = 0;
        int SliceNr1 = 0;
        SliceC::iterator SLRETURN = defineIterator(S1, SliceNr);
        SliceC::iterator SLHELP = SLRETURN;
        SliceC::Nachfolger::iterator BEGIN = SLRETURN->second.begin();
        SliceC::Nachfolger::iterator END = SLRETURN->second.end();

```

66

```

5      while (BEGIN != END) {
        if (BEGIN->second == KFK) {
          helpNr1 = BEGIN->first;
          SliceNr1 = S1[helpNr1].first.getKnotenNummer();
          SLHELP = defineIterator(S1, SliceNr1);
          SLHELP = findLastPUse2(S1, SliceNr1, RefNr);
          if (SLHELP->first.getKnotenNummer() != RefNr) {
            SLRETURN = SLHELP;
          }
        }
        BEGIN++;
      }
      return SLRETURN;
15  }

SliceC::iterator uwggraphC::lookForNextPUse(SliceC & S1, SliceC::iterator SNODE, SliceC::iterator SLPUSE) {
20      int helpNr1 = 0;
      int helpNr2 = 0;
      int helpNr3 = 0;
      int dummy = 0;
      int dummy1 = 0;
25      int KnotenNr1 = 0;
      int KnotenNr2 = 0;
      int RefNr = SLPUSE->first.getKnotenNummer();
      SliceC::iterator HELP = SNODE;
      SliceC::iterator RETURN = SNODE;
30      SliceC::Nachfolger::iterator BEGIN1 = SNODE->second.begin();
      SliceC::Nachfolger::iterator END1 = SNODE->second.end();
      while ((BEGIN1 != END1) && (dummy != 1)) {
        helpNr1 = BEGIN1->first;
        SliceC::Nachfolger::iterator BEGIN2 = S1[helpNr1].second.begin();
        SliceC::Nachfolger::iterator END2 = S1[helpNr1].second.end();
35      while ((BEGIN2 != END2) && (dummy1 != 1)) {
        if (BEGIN2->second == KFK) {
          helpNr2 = BEGIN2->first;
          if (S1[helpNr2].first.getKnotenNummer() != RefNr) {
            KnotenNr1 = S1[helpNr2].first.getKnotenNummer();
            RETURN = defineIterator(S1, KnotenNr1);
            SliceC::Nachfolger::iterator BEGIN3 = RETURN->second.begin();
40      SliceC::Nachfolger::iterator END3 = RETURN->second.end();
            while (BEGIN3 != END3) {
              if (BEGIN3->second == KFK) {
                helpNr3 = BEGIN3->first;
                if (S1[helpNr3].first.getKnotenNummer()
50      != RefNr) {
                  KnotenNr2 =
                    S1[helpNr3].first.getKnotenNummer();
                    RETURN = findLastPUse2(S1, KnotenNr2, RefNr);
55      }
                }
                BEGIN3++;
              }
              dummy1 = 1;
              dummy = 1;
            }
            else {
              KnotenNr1 = S1[helpNr1].first.getKnotenNummer();
              HELP = defineIterator(S1, KnotenNr1);
              RETURN = lookForNextPUse(S1, HELP, SLPUSE);
65      }
            }
          }
          BEGIN2++;
        }
        BEGIN1++;
70      }
      return RETURN;
    }
  }

SliceC::iterator uwggraphC::returnCondNode(SliceC & S1, int SliceNr) {
75      int helpNr = 0;

```

```

int dummy = 0;
int dummy2 = 0;
int SliceNr2 = 0;
5 SliceC::iterator SL1 = defineIterator(S1, SliceNr);
SliceC::iterator HELP = SL1;
SliceC::Nachfolger::iterator BEGIN = (*SL1).second.begin();
SliceC::Nachfolger::iterator END = (*SL1).second.end();
while ((BEGIN != END) && (dummy != 1)) {
10     helpNr = BEGIN->first;
    SliceNr = S1[helpNr].first.getKnotenNumber();
    HELP = defineIterator(S1, SliceNr);
    SliceC::Nachfolger::iterator BEGIN2 = (*HELP).second.begin();
    SliceC::Nachfolger::iterator END2 = (*HELP).second.end();
15     while ((BEGIN2 != END2) && (dummy2 != 1)) {
        if ((*BEGIN2).second == KFK) {
            helpNr = (*BEGIN2).first;
            SliceNr2 = S1[helpNr].first.getKnotenNumber();
            HELP = defineIterator(S1, SliceNr2);
20             dummy2 = 1;
            dummy = 1;
        }
        BEGIN2++;
    }
    BEGIN++;
25 }
return HELP;
}

30 //Funktion untersucht, ob zwei Kontrollstrukturen geschachtelt sind oder nicht.
int uwggraphC::checkPUses1(SliceC & S1, SliceC::iterator SLPUSEREF, SliceC::iterator
SLPUSE) {
    int helpreturn = 0;
    int KnotenNummerRef = 0;
35     int helpNr1 = 0;
    int KnotenNr1 = 0;
    KnotenNummerRef = SLPUSE->first.getKnotenNumber();
    SliceC::Nachfolger::iterator BEGIN1 = SLPUSEREF->second.begin();
    SliceC::Nachfolger::iterator END1 = SLPUSEREF->second.end();
40     while (BEGIN1 != END1) {
        if (BEGIN1->second == KFK) {
            helpNr1 = BEGIN1->first;
            KnotenNr1 = S1[helpNr1].first.getKnotenNumber();
            if (KnotenNr1 == KnotenNummerRef) {
45                 helpreturn = 1;
            }
        }
        else {
            SLPUSEREF = defineIterator(S1, KnotenNr1);
            helpreturn = checkPUses1(S1, SLPUSEREF, SLPUSE);
50        }
    }
    BEGIN1++;
}
55 return helpreturn;
}

60 int uwggraphC::checkPUses2(SliceC & S1, SliceC::iterator SLPUSEREF, SliceC::iterator
SLPUSE) {
    int helpreturn = 0;
    int KnotenNummerRef = 0;
    int helpNr1 = 0;
    int KnotenNr1 = 0;
65     KnotenNummerRef = SLPUSEREF->first.getKnotenNumber();
    SliceC::Nachfolger::iterator BEGIN1 = SLPUSE->second.begin();
    SliceC::Nachfolger::iterator END1 = SLPUSE->second.end();
    while (BEGIN1 != END1) {
        if (BEGIN1->second == KFK) {
70             helpNr1 = BEGIN1->first;
            KnotenNr1 = S1[helpNr1].first.getKnotenNumber();
            if (KnotenNr1 == KnotenNummerRef) {
                helpreturn = 1;
            }
        }
        else {
75             SLPUSE = defineIterator(S1, KnotenNr1);
            helpreturn = checkPUses2(S1, SLPUSEREF, SLPUSE);
        }
    }
}

```

```

        }
        BEGIN1++;
    }
    return helpreturn;
5  }

int uwggraphC::buildInIfTree(SliceC & S1, SliceC::iterator SLIF, SliceC::iterator SLORIG,
10 int pos1) {
    int pos2 = 0;
    int posIFFOR = 0;
    int posIFDFOR = 0;
    int posIFKFOR = 0;
15    int posIFDFORA2 = 0;
    char NodeText[128];
    char Bemerkung[1000];
    int KnotenNr = SLIF->first.getKnotenNummer();
    int LineNr = SLIF->first.getLineNr();
20    sprintf(Bemerkung, "Verzweigung(%d)", LineNr);
    strcpy(NodeText, SLIF->first.getCodeLine());
    uwgknotenC hknoten1(OR, NodeText, Bemerkung);
    pos2 = insert(hknoten1);
    posIFFOR = size() - 1;
25    verbindeEcken(pos1, pos2, 0);
    sprintf(NodeText, "Verzweigung_KF(%d)", LineNr);
    uwgknotenC hknoten5(OR, NodeText);
    posIFKFOR = insert(hknoten5);
    verbindeEcken(pos2, posIFKFOR, 0);
30    build_IFKF_Part(S1, posIFKFOR, SLIF);
    switch (SLORIG->first.getKnotenIdent()) {
        case THEN:
            {
35                sprintf(NodeText, "Verzweigung_DF_Alt.1(%d)", LineNr);
                uwgknotenC hknoten2(AND, NodeText);
                insert(hknoten1, hknoten2, 0);
                sprintf(Bemerkung, "Alt.1(%d)", LineNr);
                sprintf(NodeText, "Alt.1(%d)", LineNr);
40                uwgknotenC hknoten3(CAUSE, NodeText, Bemerkung);
                insert(hknoten2, hknoten3, 0);
                sprintf(NodeText, "Alt.1(%d)", LineNr);
                uwgknotenC hknoten4(OR, NodeText);
                insert(hknoten2, hknoten4, 0);
45                posIFDFOR = size() - 1;
                build_A1_Part(S1, posIFDFOR, SLIF, SLORIG);
            }
            break;
        case ELSE:
50            {
                sprintf(NodeText, "Verzweigung_DF_Alt.2(%d)", LineNr);
                uwgknotenC hknoten2(AND, NodeText);
                insert(hknoten1, hknoten2, 0);
                sprintf(Bemerkung, "Alt.2(%d)", LineNr);
                sprintf(NodeText, "Alt.2(%d)", LineNr);
55                uwgknotenC hknoten3(CAUSE, NodeText, Bemerkung);
                insert(hknoten2, hknoten3, 0);
                sprintf(NodeText, "Alt.2(%d)", LineNr);
                uwgknotenC hknoten4(OR, NodeText);
                insert(hknoten2, hknoten4, 0);
60                posIFDFORA2 = size() - 1;
                build_A1_Part(S1, posIFDFORA2, SLIF, SLORIG);
            }
            break;
        default:
65            cout << "Autsch!!!" << endl;
    }
    return posIFDFOR;
}

70

void uwggraphC::build_A1_Part(SliceC & S1, int posIFDFOR, SliceC::iterator SLIF, Sli-
75 ceC::iterator SLNODE) {
    int pos0 = 0;
    int pos1 = 0;
    int ret = 0;
    int dummy = 0;
    int checkNr = 1;

```

```

int checkPUse = 1;
int helpNr1 = 0;
int helpNr2 = 0;
int KnotenNr0 = 0;
5 int KnotenNr1 = 0;
int KnotenNr2 = 0;
int LineNr0 = 0;
int LineNr1 = 0;
10 char NodeText0[128];
char Bemerkung0[1000];
KnotenNr0 = SLNODE->first.getKnotenNummer();
LineNr0 = SLNODE->first.getLineNr();
sprintf(NodeText0,"Op%d",KnotenNr0);
15 //sprintf(Bemerkung0,"Line %d",LineNr0);
strcpy(Bemerkung0,SLNODE->first.getCodeLine());
uwgknotenC hknoten0(CAUSE,KnotenNr0,NodeText0,Bemerkung0);
pos0 = insert(hknoten0);
verbindeEcken(posIFDFOR,pos0,0);
20 SliceC::Nachfolger::iterator BEGIN0 = SLNODE->second.begin();
SliceC::Nachfolger::iterator END0 = SLNODE->second.end();
while (BEGIN0 != END0) {
    if (BEGIN0->second == DFK) {
        helpNr1 = BEGIN0->first;
        KnotenNr1 = S1[helpNr1].first.getKnotenNummer();
        SLNODE = defineIterator(S1,KnotenNr1);
        if (KnotenNr1 < KnotenNr0) {
            SliceC::Nachfolger::iterator BEGIN1 = SLNODE->second.begin();
            SliceC::Nachfolger::iterator END1 = SLNODE->second.end();
            while (BEGIN1 != END1) {
                if (BEGIN1->second == KFK) {
                    helpNr2 = BEGIN1->first;
                    KnotenNr2 =
30 S1[helpNr2].first.getKnotenNummer();
                    SliceC::iterator HELPPUSE = defineItera-
35 tor(S1,KnotenNr2);
                    checkNr = checkUWGNODE(KnotenNr2);
                    if (SLIF->first.getKnotenNummer() > KnotenNr2)
                        checkPUse = checkPU-
40 {
                        }
                        else {
                            checkPUse = checkPU-
45 {
                            if (KnotenNr2 != SLIF->first.getKnotenNummer())
                                if (checkPUse == 0) {
                                    dummy = 1;
                                    HELPPUSE = findOutmostPU-
50 switch (HELPPUSE-
                                    case LC:
                                        buildInLoopTree(S1, po-
                                        break;
                                    case IFC:
                                        ret = buildInIfTree(S1,
                                        break;
                                    default:
                                        cout << "Falscher Weg in
65 Al! " << endl;
                                }
                                else {
                                    if (checkNr == 0) {
                                        dummy = 1;
                                        HELPPUSE = findOutmost-
70 switch (HELPPUSE-
                                        case LC:
                                            break;
                                        case IFC:
75 buildInLoopTree(S1, posIFDFOR, HELPPUSE, SLNODE);

```



```

    int NodeNrD2 = 0;
    int KnotenNr = 0;
    int LineNr = 0;
5   KnotenNr = SLPUSE->first.getKnotenNummer();
    LineNr = SLPUSE->first.getLineNr();
    char NodeText[128];
    char Bemerkung[1000];
10   sprintf(Bemerkung,"Schleife(%d)",KnotenNr);
    strcpy(NodeText,SLPUSE->first.getCodeLine());
    uwgknotenC hknoten1(OR,NodeText,Bemerkung);
    pos1 = insert(hknoten1);
    verbindeEcken(pos1,pos1,0);
    sprintf(NodeText,"Schleife_DF_Durchl.1(%d)",LineNr);
15   uwgknotenC hknoten2(AND,NodeText);
    sprintf(NodeText,"Schleife_DF_Durchl.+(%d)",LineNr);
    uwgknotenC hknoten3(AND,NodeText);
    pos3 = insert(hknoten3);
    verbindeEcken(pos1,pos3,0);
    sprintf(NodeText,"Schleife_KF(%d)",LineNr);
20   uwgknotenC hknoten4(OR,NodeText);
    insert(hknoten1,hknoten2,0);
    insert(hknoten1,hknoten4,0);
    posLOOP_KF = size() - 1;
    sprintf(Bemerkung,"Durchl.1(%d)",LineNr);
25   sprintf(NodeText,"Durchl.1(%d)",LineNr);
    uwgknotenC hknoten5(CAUSE,NodeText,Bemerkung);
    sprintf(NodeText,"Durchl.1(%d)",LineNr);
    uwgknotenC hknoten6(OR,NodeText);
    insert(hknoten2,hknoten5,0);
30   insert(hknoten2,hknoten6,0);
    posOR_D1 = size() - 1;
    sprintf(Bemerkung,"Durchl.+(%d)",LineNr);
    sprintf(NodeText,"Durchl.+(%d)",LineNr);
35   uwgknotenC hknoten7(CAUSE,NodeText,Bemerkung);
    sprintf(NodeText,"Durchl.+(%d)",LineNr);
    uwgknotenC hknoten8(OR,NodeText);
    insert(hknoten3,hknoten7,0);
    posOR_D2 = insert(hknoten8);
    verbindeEcken(pos3, posOR_D2,0);
40   sprintf(NodeText,"KF(%d)",LineNr);
    uwgknotenC hknoten9(AND,NodeText);
    insert(hknoten4,hknoten9,0);
    insert(hknoten3,hknoten5,0);
    posAND_KF = size() - 1;
45   insert(hknoten9,hknoten5,0);
    NodeNrD2 = buildIn_LoopKF_ANDPart(S1, posAND_KF, SLPUSE);
    buildIn_LoopKF_ORPart(S1, posLOOP_KF, SLPUSE, NodeNrD2);
    //buildIn_D2_Part(S1, posOR_D2, SLPUSE, SLNODE, NodeNrD2);
    buildIn_D1_Part(S1, posOR_D1, SLPUSE, SLNODE, NodeNrD2);
50   buildIn_D2_Part(S1, posOR_D2, SLPUSE, SLNODE, NodeNrD2);
}

55 void uwggraphC::buildIn_D1_Part(SliceC & S1, int posOR_D1, SliceC::iterator SLPUSE, SliceC::iterator SLNODE, int D2_RefNr) {
    int pos1 = 0;
    int ret = 0;
    int KnotenNr1 = 0;
    int LineNr1 = 0;
60   SliceC::iterator ITER1 = find_D1_Node(S1, SLPUSE, SLNODE);
    SliceC::iterator HELP = ITER1;
    switch(ITER1->first.getKFGNodeType()) {
    case IFC:
65       HELP = find_D1_Node(S1, ITER1, SLNODE);
        ret = buildInIfTree(S1,ITER1,HELP,posOR_D1);
        break;
    case LC:
70       HELP = find_D1_Node(S1, ITER1, SLNODE);
        buildInLoopTree(S1,posOR_D1,ITER1,SLNODE);
        break;
    default:
75       addAllD1Nodes(S1,ITER1,posOR_D1,D2_RefNr);
    }
}

```

```

SliceC::iterator uwggraphC::find_D1_Node(SliceC & S1, SliceC::iterator SLPUSE, SliceC::iterator SLNODE) {
    int helpNr1 = 0;
    int helpNr2 = 0;
5    int dummy1 = 0;
    int dummy2 = 0;
    int KnotenNrRet = 0;
    int KnotenNrRet1 = 0;
10    int PUseNr = SLPUSE->first.getKnotenNummer();
    int PUseLevel = SLPUSE->first.getLevel();
    SliceC::iterator RETURNITER = SLNODE;
    KnotenNrRet1 = SLNODE->first.getKnotenNummer();
    if (KnotenNrRet1 == (PUseNr + 1)) {
15        RETURNITER = defineIterator(S1, KnotenNrRet1);
        dummy1 = 1;
        dummy2 = 1;
    }
    SliceC::Nachfolger::iterator BEGIN1 = SLNODE->second.begin();
    SliceC::Nachfolger::iterator END1 = SLNODE->second.end();
20    while ((BEGIN1 != END1) && (dummy1 != 1)) {
        helpNr1 = BEGIN1->first;
        if ((S1[helpNr1].first.getKnotenNummer() > PUseNr) &&
            (S1[helpNr1].first.getLevel() >= PUseLevel)) {
25            if (S1[helpNr1].first.getLevel() > PUseLevel) {
                KnotenNrRet1 = S1[helpNr1].first.getKnotenNummer();
                if (KnotenNrRet1 == (PUseNr + 1)) {
                    RETURNITER = defineIterator(S1, KnotenNrRet1);
                    dummy1 = 1;
                    dummy2 = 1;
30                }
                SliceC::Nachfolger::iterator BEGIN2 =
                    SliceC::Nachfolger::iterator END2 = S1[helpNr1].second.end();
                while ((BEGIN2 != END2) && (dummy2 != 1)) {
35                    if (BEGIN2->second == KFK) {
                        helpNr2 = BEGIN2->first;
                        KnotenNrRet =
                            S1[helpNr2].first.getKnotenNummer();
40                        if (KnotenNrRet != PUseNr) {
                            RETURNITER = defineItera-
                                tor(S1, KnotenNrRet);
                                dummy1 = 1;
                                dummy2 = 1;
45                        }
                    }
                    BEGIN2++;
                }
            }
            else {
50                KnotenNrRet = S1[helpNr1].first.getKnotenNummer();
                RETURNITER = defineIterator(S1, KnotenNrRet);
                dummy1 = 1;
            }
        }
        BEGIN1++;
55    }
    if ((RETURNITER == SLNODE) && (dummy1 == 0)) {
        BEGIN1 = SLNODE->second.begin();
        while (BEGIN1 != END1) {
60            helpNr1 = BEGIN1->first;
            KnotenNrRet = S1[helpNr1].first.getKnotenNummer();
            SliceC::iterator SLNODE = defineIterator(S1, KnotenNrRet);
            RETURNITER = find_D1_Node(S1, SLPUSE, SLNODE);
            BEGIN1++;
65        }
    }
    return RETURNITER;
70
}

void uwggraphC::buildIn_D2_Part(SliceC & S1, int posOR_D2, SliceC::iterator SLPUSE, SliceC::iterator SLNODE, int KnotenNrD2) {
75    int pos1 = 0;
    int pos2 = 0;
    int ret = 0;
    int helpNr1 = 0;
    int helpNr2 = 0;

```

```

int KnotenNr1 = 0;
int LineNr1 = 0;
int KnotenNrD2Ret = 0;
int USEinD2KnotenNr = 0;
5 char Bemerkung1[1000];
char NodeText1[128];
SliceC::iterator D2_NODE = defineIterator(S1, KnotenNrD2);
KnotenNr1 = D2_NODE->first.getKnotenNummer();
LineNr1 = D2_NODE->first.getLineNr();
10 sprintf(NodeText1,"Op%d",KnotenNr1);
//sprintf(Bemerkung1,"Line %d",LineNr1);
strcpy(Bemerkung1,D2_NODE->first.getCodeLine());
uwgknotenC hknoten1(CAUSE,KnotenNr1,NodeText1,Bemerkung1);
pos1 = insert(hknoten1);
15 verbindeEcken(posOR_D2,pos1,0);
SliceC::Nachfolger::iterator BEGIN1 = D2_NODE->second.begin();
SliceC::Nachfolger::iterator END1 = D2_NODE->second.end();
while (BEGIN1 != END1) {
20     if (BEGIN1->second == DFK) {
        helpNr1 = BEGIN1->first;
        KnotenNr1 = S1[helpNr1].first.getKnotenNummer();
        LineNr1 = S1[helpNr1].first.getLineNr();
        sprintf(NodeText1,"Op%d",KnotenNr1);
        //sprintf(Bemerkung1,"Line %d",LineNr1);
        strcpy(Bemerkung1,S1[helpNr1].first.getCodeLine());
        uwgknotenC hknoten2(CAUSE,KnotenNr1,NodeText1,Bemerkung1);
        pos2 = insert(hknoten2);
        verbindeEcken(posOR_D2,pos2,0);
        SliceC::Nachfolger::iterator BEGIN2 = S1[helpNr1].second.begin();
        SliceC::Nachfolger::iterator END2 = S1[helpNr1].second.end();
        while (BEGIN2 != END2) {
            helpNr2 = BEGIN2->first;
            if ((BEGIN2->second == DFK) &&
35 (S1[helpNr2].first.getKnotenNummer() != KnotenNrD2)) {
                //SliceC::iterator SLNODE1 = defineItera-
                tor(S1,KnotenNr1);
                //buildIn_D2_Part(S1,posOR,SLPUSE,SLNODE1,KnotenNrD2);
            }
            BEGIN2++;
40         }
        BEGIN1++;
    }
}
SliceC::iterator USEinD2 = find_D1_Node(S1, SLPUSE, SLNODE);
SliceC::iterator HELP = USEinD2;
45 if (USEinD2->first.getLevel() > SLPUSE->first.getLevel()) {
    USEinD2KnotenNr = USEinD2->first.getKnotenNummer();
    KnotenNrD2Ret = findD2Node(S1, USEinD2);
    if (KnotenNrD2Ret == USEinD2KnotenNr) {
50         switch(USEinD2->first.getKFGNodeType()) {
            case IFC:
                HELP = find_D1_Node(S1,USEinD2,SLNODE);
                //ret = buildInIfTree(S1,USEinD2,HELP,posOR);
                ret = buildD2InIfTree(S1,USEinD2,HELP,posOR_D2);
                break;
55             case LC:
                HELP = find_D1_Node(S1,USEinD2,SLNODE);
                buildInLoopTree(S1,posOR_D2,USEinD2,SLNODE);
                break;
60             default:
                cout << "Uups!!!" << endl;
            }
        }
    }
65 }

int uwggraphC::findD2Node(SliceC & S1, SliceC::iterator PUSE) {
70     int dummy = 0;
    int helpNr1 = 0;
    int helpNr2 = 0;
    int KnotenNrReturn = PUSE->first.getKnotenNummer();
    int PUSEKnotenNr = PUSE->first.getKnotenNummer();
    SliceC::Nachfolger::iterator BEGIN1 = PUSE->second.begin();
    SliceC::Nachfolger::iterator END1 = PUSE->second.end();
    while ((BEGIN1 != END1) && (dummy != 1)) {
75         if (BEGIN1->second == DFK) {
            helpNr1 = BEGIN1->first;

```

75

```

SliceC::Nachfolger::iterator BEGIN2 = S1[helpNr1].second.begin();
SliceC::Nachfolger::iterator END2 = S1[helpNr1].second.end();
while ((BEGIN2 != END2) && (dummy != 1)) {
    if (BEGIN2->second == KFK) {
        helpNr2 = BEGIN2->first;
        if (S1[helpNr2].first.getKnotenNummer() == PUSEKno-
5         tenNr) {
            KnotenNrReturn =
10         S1[helpNr2].first.getKnotenNummer();
            dummy = 1;
        }
        BEGIN2++;
    }
    BEGIN1++;
}
return KnotenNrReturn;
}

int uwggraphC::buildIn_LoopKF_ANDPart(SliceC & S1, int posAND_KF, SliceC::iterator SLPUSE)
{
25     int pos = 0;
    int helpNr1 = 0;
    int helpNr2 = 0;
    int RefPUse = 0;
    int KnotenNr = 0;
30     int LineNr = 0;
    int dummy = 0;
    int dummy1 = 0;
    char Bemerkung[1000];
    char NodeText[128];
35     RefPUse = SLPUSE->first.getKnotenNummer();
    SliceC::Nachfolger::iterator BEGIN = SLPUSE->second.begin();
    SliceC::Nachfolger::iterator END = SLPUSE->second.end();
    while ((BEGIN != END) && (dummy1 != 1)) {
        if (BEGIN->second == DFK) {
40             helpNr1 = BEGIN->first;
            SliceC::Nachfolger::iterator BEGIN2 = S1[helpNr1].second.begin();
            SliceC::Nachfolger::iterator END2 = S1[helpNr1].second.end();
            while ((BEGIN2 != END2) && (dummy != 1)) {
                if (BEGIN2->second == KFK) {
45                     helpNr2 = BEGIN2->first;
                    if (S1[helpNr2].first.getKnotenNummer() == RefPUse) {
                        KnotenNr = S1[helpNr1].first.getKnotenNummer();
                        LineNr = S1[helpNr1].first.getLineNr();
                        sprintf(NodeText, "Op%d", KnotenNr);
                        //sprintf(Bemerkung, "Line %d", LineNr);
50
                        strcpy(Bemerkung, S1[helpNr1].first.getCodeLine());
                        uwgknotenC hkno-
55         ten(CAUSE, KnotenNr, NodeText, Bemerkung);
                        pos = insert(hknoten);
                        verbindeEcken(posAND_KF, pos, 0);
                        dummy = 1;
                        dummy1 = 1;
                    }
                }
                BEGIN2++;
            }
            BEGIN++;
60
        }
        return KnotenNr;
    }
}

void uwggraphC::buildIn_LoopKF_ORPart(SliceC & S1, int posLOOP_KF, SliceC::iterator SL1,
int KnotenNrD2) {
70     int pos1 = 0;
    int pos2 = 0;
    int helpNr1 = 0;
    int helpNr2 = 0;
    int KnotenNr = 0;
75     int KnotenNr1 = 0;
    int LineNr = 0;

```

```

int LineNr1 = 0;
char NodeText[128];
char Bemerkung[1000];
KnotenNr = SL1->first.getKnotenNummer();
5   LineNr = SL1->first.getLineNr();
    sprintf(NodeText,"Op%d",KnotenNr);
    //sprintf(Bemerkung,"Line %d",LineNr);
    strcpy(Bemerkung,SL1->first.getCodeLine());
10   uwgknotenC hknoten1(CAUSE, KnotenNr, NodeText,Bemerkung);
    pos1 = insert(hknoten1);
    verbindeEcken(posLOOP_KF,pos1,0);
    SliceC::Nachfolger::iterator BEGIN = SL1->second.begin();
    SliceC::Nachfolger::iterator END = SL1->second.end();
15   while (BEGIN != END) {
        if (BEGIN->second == DFK) {
            helpNr1 = BEGIN->first;
            KnotenNr1 = S1[helpNr1].first.getKnotenNummer();
            if (KnotenNr1 != KnotenNrD2) {
20                 LineNr1 = S1[helpNr1].first.getLineNr();
                    sprintf(NodeText,"Op%d",KnotenNr1);
                    //sprintf(Bemerkung,"Line %d",LineNr1);
                    strcpy(Bemerkung,SL1->first.getCodeLine());
                    uwgknotenC hknoten1(CAUSE, KnotenNr1, NodeText,Bemerkung);
25                 pos1 = insert(hknoten1);
                    verbindeEcken(posLOOP_KF,pos1,0);
            }
            SliceC::iterator NEXTNODE = defineIterator(S1,KnotenNr1);
            SliceC::Nachfolger::iterator BEGIN2 = NEXTNODE->second.begin();
            SliceC::Nachfolger::iterator END2 = NEXTNODE->second.end();
30         while (BEGIN2 != END2) {
            if (BEGIN2->second == DFK) {
                helpNr2 = BEGIN2->first;
                if (NEXTNODE->first.getKnotenNummer() != KnotenNrD2) {
35                     buildIn_LoopKF_ORPart(S1, posLOOP_KF, NEXTNODE,
                        KnotenNrD2);
                }
            }
            BEGIN2++;
40         }
        BEGIN++;
    }
}

45 void uwggraphC::addAllD1Nodes(SliceC & S1, SliceC::iterator SLNODE, int posGatter, int
D2_RefNr) {
    int pos1 = 0;
    int ret = 0;
50   int checkNr = 1;
    int helpNr1 = 0;
    int helpNr2 = 0;
    int SliceNr2 = 0;
    int KnotenNr0 = 0;
55   int KnotenNr1 = 0;
    int KnotenNr2 = 0;
    int LineNr1 = 0;
    int LineNr0 = 0;
60   char Bemerkung1[1000];
    char NodeText1[128];
    KnotenNr0 = SLNODE->first.getKnotenNummer();
    LineNr0 = SLNODE->first.getLineNr();
    sprintf(NodeText1,"Op%d",KnotenNr0);
65   //sprintf(Bemerkung1,"Line %d",LineNr0);
    strcpy(Bemerkung1,SLNODE->first.getCodeLine());
    uwgknotenC hknoten1(CAUSE,KnotenNr0,NodeText1,Bemerkung1);
    pos1 = insert(hknoten1);
    verbindeEcken(posGatter,pos1,0);
70   SliceC::Nachfolger::iterator BEGIN1 = SLNODE->second.begin();
    SliceC::Nachfolger::iterator END1 = SLNODE->second.end();
    while (BEGIN1 != END1) {
        if (BEGIN1->second == DFK) {
            helpNr1 = BEGIN1->first;
            KnotenNr1 = S1[helpNr1].first.getKnotenNummer();
75         SLNODE = defineIterator(S1,KnotenNr1);
            if ((KnotenNr1 != D2_RefNr) && (KnotenNr1 != KnotenNr0)) {
                SliceC::Nachfolger::iterator BEGIN2 =
S1[helpNr1].second.begin();

```

77

```

SliceC::Nachfolger::iterator END2 = S1[helpNr1].second.end();
while (BEGIN2 != END2) {
    if (BEGIN2->second == KFK) {
        helpNr2 = BEGIN2->first;
        SliceNr2 = S1[helpNr2].first.getKnotenNumber();
        SliceC::iterator HELPPUSE = defineIterator(S1,
5          SliceNr2);

        checkNr = checkUWGNODE(SliceNr2);
        if (checkNr == 0) {
10          SliceC::iterator HELP1 = findOutmostPU-
            switch (HELP1->first.getKFGNodeType())
                case LC:
                    buildInLoopTree(S1, posGatter,
20                    break;
                case IFC:
                    ret = buildInIf-
                    break;
                default:
                    cout << "Falscher Weg! " <<
25          endl;
        }
    }
    BEGIN2++;
30  }
    if (checkNr == 1) {
        LineNr1 = S1[helpNr1].first.getLineNr();
        sprintf(NodeText1,"Op%d",KnotenNr1);
        //sprintf(Bemerkung1,"Line %d",LineNr1);
        strcpy(Bemerkung1,S1[helpNr1].first.getCodeLine());
        uwgknotenC hkno-
35      ten2(CAUSE,KnotenNr1,NodeText1,Bemerkung1);
        pos1 = insert(hknoten2);
        verbindeEcken(posGatter,pos1,0);
        addAllD1Nodes(S1,SLNODE,posGatter,D2_RefNr);
40      }
    }
    BEGIN1++;
45  }

int uwggraphC::buildD2InIfTree(SliceC & S1, SliceC::iterator SLIF, SliceC::iterator SLORIG,
int pos1) {
50    int pos2 = 0;
    int posIFOR = 0;
    int posIFDFOR = 0;
    int posIFKFOR = 0;
    int posIFDFORA2 = 0;
55    char NodeText[128];
    char Bemerkung[1000];
    int KnotenNr = SLIF->first.getKnotenNumber();
    int LineNr = SLIF->first.getLineNr();
    sprintf(NodeText,"Verzweigung_D+ (%d)",KnotenNr);
    strcpy(Bemerkung,SLIF->first.getCodeLine());
    uwgknotenC hknoten1(OR,NodeText,Bemerkung);
    pos2 = insert(hknoten1);
    posIFOR = size() - 1;
    verbindeEcken(pos1, pos2,0);
65    sprintf(NodeText,"Verzweigung_KF_D+ (%d)",LineNr);
    uwgknotenC hknoten5(OR,NodeText);
    posIFKFOR = insert(hknoten5);
    verbindeEcken(pos2,posIFKFOR,0);
    buildD2_IFKF Part(S1, posIFKFOR, SLIF);
70    switch (SLORIG->first.getKnotenIdent()) {
        case THEN:
            {
                sprintf(NodeText,"Verzweigung_DF_Alt.1_D+ (%d)",LineNr);
                uwgknotenC hknoten2(AND,NodeText);
                insert(hknoten1,hknoten2,0);
                sprintf(Bemerkung,"Alt.1_D+ (%d)",LineNr);
                sprintf(NodeText,"Alt.1_D+ (%d)",LineNr);
                uwgknotenC hknoten3(CAUSE,NodeText,Bemerkung);
75            }
    }
}

```

78

```

        insert(hknoten2,hknoten3,0);
        sprintf(NodeText,"Alt.1_D+(%d)",LineNr);
        uwgknotenC hknoten4(OR,NodeText);
        insert(hknoten2,hknoten4,0);
5      posIFDFOR = size() - 1;
        buildD2_A1_Part(S1, posIFDFOR, SLIF, SLORIG);
    }
    break;
10   case ELSE:
    {
        sprintf(NodeText,"Verzweigung_DF_Alt.2_D+(%d)",LineNr);
        uwgknotenC hknoten2(AND,NodeText);
        insert(hknoten1,hknoten2,0);
        sprintf(Bemerkung,"Alt.2_D+(%d)",LineNr);
15      sprintf(NodeText,"Alt.2_D+(%d)",LineNr);
        uwgknotenC hknoten3(CAUSE,NodeText,Bemerkung);
        insert(hknoten2,hknoten3,0);
        sprintf(NodeText,"Alt.2_D+(%d)",LineNr);
        uwgknotenC hknoten4(OR,NodeText);
20      insert(hknoten2,hknoten4,0);
        posIFDFORA2 = size() - 1;
        buildD2_A1_Part(S1, posIFDFORA2, SLIF, SLORIG);
    }
    break;
25   default:
        cout << "Autsch!!!" << endl;
    }
    return posIFDFOR;
30 }

void uwggraphC::buildD2_A1_Part(SliceC & S1, int posIFDFOR_D2, SliceC::iterator SLIF, SliceC::iterator SLNODE) {
35     int pos0_A1 = 0;
    int pos1_A1 = 0;
    int helpNr0 = 0;
    int helpNr1 = 0;
    int KnotenNr0 = 0;
    int KnotenNr1 = 0;
40     int KnotenNr2 = 0;
    int LineNr0 = 0;
    int LineNr1 = 0;
    char NodeText0[128];
    char Bemerkung0[1000];
45     KnotenNr0 = SLNODE->first.getKnotenNummer();
    LineNr0 = SLNODE->first.getLineNr();
    sprintf(NodeText0,"Op%d",KnotenNr0);
    //sprintf(Bemerkung0,"Line %d",LineNr0);
    strcpy(Bemerkung0,SLNODE->first.getCodeLine());
50     uwgknotenC hknoten0(CAUSE,KnotenNr0,NodeText0,Bemerkung0);
    pos0_A1 = insert(hknoten0);
    verbindeEcken(posIFDFOR_D2,pos0_A1,0);
    SliceC::Nachfolger::iterator BEGIN0 = SLNODE->second.begin();
    SliceC::Nachfolger::iterator END0 = SLNODE->second.end();
55     while (BEGIN0 != END0) {
        helpNr0 = BEGIN0->first;
        KnotenNr1 = S1[helpNr0].first.getKnotenNummer();
        if (KnotenNr1 > KnotenNr0) {
            LineNr1 = S1[helpNr0].first.getLineNr();
60             sprintf(NodeText0,"Op%d",KnotenNr1);
            //sprintf(Bemerkung0,"Line %d",LineNr1);
            strcpy(Bemerkung0,S1[helpNr0].first.getCodeLine());
            uwgknotenC hknoten1(CAUSE,KnotenNr1,NodeText0,Bemerkung0);
            pos1_A1 = insert(hknoten1);
65             verbindeEcken(posIFDFOR_D2,pos1_A1,0);
            SliceC::Nachfolger::iterator BEGIN1 = S1[helpNr0].second.begin();
            SliceC::Nachfolger::iterator END1 = S1[helpNr0].second.end();
            while (BEGIN1 != END1) {
                if (BEGIN1->second == DFK) {
70                     helpNr1 = BEGIN1->first;
                    KnotenNr2 = S1[helpNr1].first.getKnotenNummer();
                    if (KnotenNr1 != KnotenNr2) {
                        SliceC::iterator SLNODE = defineIterator(S1,KnotenNr2);
75                     buildD2_A1_Part(S1, posIFDFOR_D2, SLIF,
                        SLNODE);
                    }
                }
            }
        }
    }
}

```



```

WNode(KnotenNr2);

5      dOutmostPUse(S1,HELPPUSE);
      >first.getKFGNodeType()) {

10          buildInLoopTree(S1, posKFOR_D2, HELPPUSE, SLNODE);

15      buildInIfTree(S1, HELPPUSE, SLNODE, posKFOR_D2);

20      "Falscher Weg in A1! " << endl;

25      S1[helpNr2].first.getLineNr();

          sprintf(NodeText0,"Op%d",KnotenNr2);

30          //sprintf(Bemerkung0,"Line %d",LineNr1);

          strcpy(Bemerkung0,S1[helpNr2].first.getCodeLine());

35      tenl(CAUSE,KnotenNr2,NodeText0,Bemerkung0);

          insert(hknoten1);

          verbindeEcken(posKFOR_D2,pos1_D2,0);

40      }

          }

          BEGIN2++;

45      }

          BEGIN1++;

          }

          BEGIN0++;

50      }

SliceC::iterator uwggraphC::findOutmostPUse(SliceC & S1, SliceC::iterator SLPUSE) {
55      int dummy = 0;
      int helpNr1 = 0;
      int checkNr = 0;
      int KnotenNr1 = 0;
      SliceC::iterator RETURN = SLPUSE;
60      SliceC::Nachfolger::iterator BEGIN1 = SLPUSE->second.begin();
      SliceC::Nachfolger::iterator END1 = SLPUSE->second.end();
      while ((BEGIN1 != END1) && (dummy != 1)){
          if (BEGIN1->second == KFK) {
70              helpNr1 = BEGIN1->first;
              KnotenNr1 = S1[helpNr1].first.getKnotenNummer();
              checkNr = checkUWGNode(KnotenNr1);
              if (checkNr == 0) {
                  SLPUSE = defineIterator(S1,KnotenNr1);
                  RETURN = findOutmostPUse(S1, SLPUSE);
                  dummy = 1;
              }
              else {
75              RETURN = SLPUSE;
                  dummy = 1;
              }
          }
          BEGIN1++;
      }
}

```

checkNode = checkU-

```

if (checkNode == 0) {
    if (checkPUse == 0) {
        //HELPPUSE = fin-

```

```

        switch (HELPPUSE-
            case LC:

```

```

                dummy = 1;
                break;
            case IFC:
                ret =

```

```

                dummy = 1;
                break;
            default:
                cout <<

```

```

        }
    }
}

```

```

else {
    LineNr1 =

```

uwgknotenC hkno-

pos1_D2 =

```

        return RETURN;
    }

5   int uwggraphC::checkUWGNode(int SliceNr) {
        int dummy = 0;
        uwggraphC::iterator ITER = begin();
        while (ITER != end()) {
10          if ((*ITER).first.getCauseNr() == SliceNr) {
                dummy = 1;
            }
            ITER++;
        }
        if (dummy == 1)
15          return 1;
        else
            return 0;
    }

20   SliceC::iterator uwggraphC::defineIterator(SliceC & S1, int SliceNr) {
        int dummy = 0;
        SliceC::iterator ITER = S1.begin();
        SliceC::iterator HELP = S1.begin();
25        while ((ITER != S1.end()) && (dummy != 1)) {
            if ((*ITER).first.getKnotenNummer() == SliceNr) {
                HELP = ITER;
                dummy = 1;
            }
            ITER++;
30        }
        return HELP;
    }

35   void uwggraphC::SliceAusgeben(SliceC & S1) {
        //int a;
        ofstream Ziel("Slice.slc");
40        ostream_iterator<KFGLListNodeC> POSIT(Ziel, "Knoten\n");
        ostream_iterator<KFGLListNodeC> POSIT2(Ziel, "Nachfolger: ");
        ostream_iterator<GraphKanteC> KANTEIT(Ziel, "\n");
        Ziel << "SLICE:" << endl << endl;
        for (int i = 0; i < S1.size(); ++i) {
45            Ziel << S1[i].first << " <";
            SliceC::Nachfolger::iterator IT = S1[i].second.begin();
            SliceC::Nachfolger::iterator ITEND = S1[i].second.end();
            while (IT != ITEND) {
50                Ziel << S1[(*IT).first].first << ' ' // Ecke
                << endl << "Kante:" << (*IT).second << ' '; // Kantenwert
                ++IT;
            }
            Ziel << ">\n";
55        }
    }

60

#include "uwgkante.h"
65 #include <iostream>

    ostream& operator << (ostream& os, const uwgkanteC& Node){
70     os << Node.Number << endl ;
        return os;
75 }

```

```

#include "uwgknoten.h"
□
#include <iostream>
□
5
□
ostream& operator << ( ostream& os, const uwgknotenC& Node){
    //os << Node.GatterIdent << " " << Node.CauseNr << " " << Node.gattertext << endl
10
    ;
    //return os;
    if (Node.getNodeIdent() == EFFECT) {
        os << "%%EFFECT:\n" << Node.gatterbemerkung << "\n:\n" << Node.gattertext <<
"\n," << Node.GatterIdent << ",, /0;" << endl;
15
        return os;
    }
    else if (Node.getNodeIdent() == OR) {
        os << "%%OR:1:\n" << Node.gattertext << "\n:\nkeine\n," << Node.GatterIdent
<< ",, /0;" << endl;
20
        return os;
    }
    else if (Node.getNodeIdent() == AND) {
        os << "%%AND:\n" << Node.gattertext << "\n:\nkeine\n," << Node.GatterIdent
<< ",, /0;" << endl;
25
        return os;
    }
    else if (Node.getNodeIdent() == NOT) {
        os << "%%NOT:\n" << Node.gattertext << "\n:\nkeine\n," << Node.GatterIdent
<< ",, /0;" << endl;
30
        return os;
    }
    /*else if (Node.getNodeIdent() == CAUSE) {
        os << "%%CAUSE:\n" << Node.gattertext << "\n:\nkeine\n," << Node.GatterIdent
<< ",, /0;" << endl;
35
        return os;
    }*/
    else if (Node.getNodeIdent() == CAUSE) {
        os << "%%CAUSE:\n" << Node.gatterbemerkung << "\n:\n" << Node.gattertext <<
"\n," << Node.GatterIdent << ",, /0;" << endl;
40
        return os;
    }
    else {
        os << Node.GatterIdent << " " << Node.CauseNr << " " << Node.gattertext
<< endl ;
45
        return os;
    }
}

int uwgknotenC::DummyNr = 1;

```

In diesem Dokument sind folgende Veröffentlichungen zitiert:

- [1] N. Leveson, Safety Verification of ADA Programs using software fault trees, IEEE Software, Seite 48 bis 59, Juli 5 1991
- [2] Weiser M., *Program Slicing*, in: IEEE Transaction on Software Engineering, Vol. 10, No. 4, July 1984, pp. 352-357
- 10 [3] Liggesmeyer P., Modultest und Modulverifikation - State of the Art, Mannheim, Wien, Zürich: BI Wissenschaftsverlag 1990
- [4] DIN 25424-1: Fehlerbaumanalysen; Methoden und Bildzei- 15 chen, September 1981
- [5] DIN 25424-2: Fehlerbaumanalyse; Handrechenverfahren zur Auswertung eines Fehlerbaums, Berlin, Beuth Verlag GmbH, April 1990

Patentansprüche

1. Verfahren zur Ermittlung einer Gesamtfehlerbeschreibung
zumindest eines Teils eines Computerprogramms, durch einen
5 Computer,
 - bei dem zumindest der Teil des Computerprogramms gespeichert ist,
 - bei dem eine Kontrollflußbeschreibung und eine Datenflußbeschreibung für den Teil des Computerprogramms ermittelt
10 werden,
 - bei dem Programmelemente aus dem Teil des Computerprogramms ausgewählt werden,
 - bei dem für jedes ausgewählte Programmelement unter Verwendung einer gespeicherten Fehlerbeschreibung, die jeweils
15 einem Referenzelement zugeordnet ist, eine Elementenfehlerbeschreibung ermittelt wird, mit der mögliche Fehler des jeweiligen Programmelements beschrieben werden,
 - bei dem mit einer Fehlerbeschreibung eines Referenzelements mögliche Fehler des jeweiligen Referenzelements beschrieben
20 werden, und
 - bei dem aus den Elementenfehlerbeschreibungen die Gesamtfehlerbeschreibung unter Berücksichtigung der Kontrollflußbeschreibung und der Datenflußbeschreibung ermittelt wird.
- 25 2. Verfahren nach Anspruch 1,
bei dem die Kontrollflußbeschreibung in Form eines Kontrollflußgraphen vorliegt.
- 30 3. Verfahren nach Anspruch 1 oder 2,
bei dem die Datenflußbeschreibung in Form eines Datenflußgraphen vorliegt.
4. Verfahren nach einem der vorangegangenen Ansprüche,
35 • bei dem die Fehlerbeschreibung in Form eines gespeicherten Fehlerbaums vorliegt,

- bei dem die Elementenfehlerbeschreibung als Elementenfehlerbaum ermittelt wird, und
- bei dem die Gesamtfehlerbeschreibung als Gesamtfehlerbaum ermittelt wird.

5

5. Verfahren nach einem der vorangegangenen Ansprüche, eingesetzt zur Fehleranalyse des Teils des Computerprogramms.

6. Verfahren nach einem der vorangegangenen Ansprüche,

- 10
- bei dem die Gesamtfehlerbeschreibung als Gesamtfehlerbaum ermittelt wird,
 - bei dem der Gesamtfehlerbaum verändert wird hinsichtlich vorgegebener Rahmenbedingungen.

15 7. Verfahren nach Anspruch 6,

bei dem die Veränderung durch Hinzufügen eines Ergänzungsfehlerbaums erfolgt.

8. Anordnung zur Ermittlung einer Gesamtfehlerbeschreibung
20 zumindest eines Teils eines Computerprogramms, durch einen Computer,

mit einem Prozessor, der derart eingerichtet ist, daß folgende Schritte durchführbar sind:

- zumindest der Teil des Computerprogramms ist gespeichert,
- 25 • eine Kontrollflußbeschreibung und eine Datenflußbeschreibung werden für den Teil des Computerprogramms ermittelt,
- Programmelemente werden aus dem Teil des Computerprogramms ausgewählt,
- für jedes ausgewählte Programmelement wird unter Verwendung
30 einer gespeicherten Fehlerbeschreibung, die jeweils einem Referenzelement zugeordnet ist, eine Elementenfehlerbeschreibung ermittelt, mit der mögliche Fehler des jeweiligen Programmelements beschrieben werden,
- mit einer Fehlerbeschreibung eines Referenzelements werden
35 mögliche Fehler des jeweiligen Referenzelements beschrieben, und

- aus den Elementenfehlerbeschreibungen wird die Gesamtfehlerbeschreibung unter Berücksichtigung der Kontrollflußbeschreibung und der Datenflußbeschreibung ermittelt.
- 5 9. Anordnung nach Anspruch 8,
bei der der Prozessor derart eingerichtet ist, daß die Kontrollflußbeschreibung in Form eines Kontrollflußgraphen vorliegt.
- 10 10. Anordnung nach Anspruch 8 oder 9,
bei der der Prozessor derart eingerichtet ist, daß die Datenflußbeschreibung in Form eines Datenflußgraphen vorliegt.
11. Anordnung nach einem der Ansprüche 8 bis 10,
15 bei der der Prozessor derart eingerichtet ist, daß
- die Fehlerbeschreibung in Form eines gespeicherten Fehlerbaums vorliegt,
 - die Elementenfehlerbeschreibung als Elementenfehlerbaum ermittelt wird, und
 - 20 • die Gesamtfehlerbeschreibung als Gesamtfehlerbaum ermittelt wird.
12. Anordnung nach einem der Ansprüche 8 bis 11,
eingesetzt zur Fehleranalyse des Teils des Computerprogramms.
- 25 13. Anordnung nach einem der Ansprüche 8 bis 12,
bei der der Prozessor derart eingerichtet ist, daß
- die Gesamtfehlerbeschreibung als Gesamtfehlerbaum ermittelt wird,
 - 30 • der Gesamtfehlerbaum verändert wird hinsichtlich vorgegebener Rahmenbedingungen.
14. Anordnung nach Anspruch 13,
bei der der Prozessor derart eingerichtet ist, daß die Veränderung durch Hinzufügen eines Ergänzungsfehlerbaums erfolgt.
- 35

15. Computerprogramm-Erzeugnis, das ein computerlesbares Speichermedium umfaßt, auf dem ein Programm gespeichert ist, das es einem Computer ermöglicht, nachdem es in einen Speicher des Computers geladen worden ist, folgende Schritte durchzuführen zur Ermittlung einer Gesamtfehlerbeschreibung zumindest eines Teils eines Computerprogramms:
- 5 • zumindest der Teil des Computerprogramms ist gespeichert,
 - eine Kontrollflußbeschreibung und eine Datenflußbeschreibung werden für den Teil des Computerprogramms ermittelt,
 - 10 • Programmelemente werden aus dem Teil des Computerprogramms ausgewählt,
 - für jedes ausgewählte Programmelement wird unter Verwendung einer gespeicherten Fehlerbeschreibung, die jeweils einem Referenzelement zugeordnet ist, eine Elementenfehlerbe-
 - 15 schreibung ermittelt, mit der mögliche Fehler des jeweiligen Programmelements beschrieben werden,
 - mit einer Fehlerbeschreibung eines Referenzelements werden mögliche Fehler des jeweiligen Referenzelements beschrieben, und
 - 20 • aus den Elementenfehlerbeschreibungen wird die Gesamtfehlerbeschreibung unter Berücksichtigung der Kontrollflußbeschreibung und der Datenflußbeschreibung ermittelt.
16. Computerlesbares Speichermedium, auf dem ein Programm gespeichert ist, das es einem Computer ermöglicht, nachdem es in einen Speicher des Computers geladen worden ist, folgende Schritte durchzuführen zur Ermittlung einer Gesamtfehlerbeschreibung zumindest eines Teils eines Computerprogramms:
- 25 • zumindest der Teil des Computerprogramms ist gespeichert,
 - 30 • eine Kontrollflußbeschreibung und eine Datenflußbeschreibung werden für den Teil des Computerprogramms ermittelt,
 - Programmelemente werden aus dem Teil des Computerprogramms ausgewählt,
 - für jedes ausgewählte Programmelement wird unter Verwendung
 - 35 einer gespeicherten Fehlerbeschreibung, die jeweils einem Referenzelement zugeordnet ist, eine Elementenfehlerbe-

schreibung ermittelt, mit der mögliche Fehler des jeweiligen Programmelements beschrieben werden,

- mit einer Fehlerbeschreibung eines Referenzelements werden mögliche Fehler des jeweiligen Referenzelements beschrieben, und
- 5 • aus den Elementenfehlerbeschreibungen wird die Gesamtfehlerbeschreibung unter Berücksichtigung der Kontrollflußbeschreibung und der Datenflußbeschreibung ermittelt.

Zusammenfassung

**Verfahren und Anordnung zur Ermittlung einer Gesamtfehlerbe-
schreibung zumindest eines Teils eines Computerprogramms so-
wie Computerprogramm-Erzeugnis und computerlesbares Speicher-
medium**

Aus dem Teil des Computerprogramms werden eine Kontrollfluß-
beschreibung und eine Datenflußbeschreibung ermittelt und es
werden Programmelemente aus dem Teil des Computerprogramms
ausgewählt. Für jedes ausgewählte Programmelement wird unter
Verwendung einer gespeicherten Fehlerbeschreibung die jeweils
einem Referenzelement zugeordnet ist, eine Elementenfehlerbe-
schreibung ermittelt, mit der mögliche Fehler des jeweiligen
Programmelements beschrieben werden. Aus den Elementenfehler-
beschreibungen wird die Gesamtfehlerbeschreibung unter Be-
rücksichtigung der Kontrollflußbeschreibung und der Daten-
flußbeschreibung ermittelt.

Figur 2